ADO .NET

Most applications need data access at one point of time making it a crucial component when working with applications. Data access is making the application interact with a database, where all the data is stored. Different applications have different requirements for database access. ASP .NET uses ADO .NET (Active X Data Object) as it's data access and manipulation protocol which also enables us to work with data on the Internet. Let's take a look why ADO .NET came into picture replacing ADO.

What is ADO.NET?

ADO.NET is not a different technology. In simple terms, you can think of ADO.NET, as a set of classes (Framework), which can be used to interact with data sources like Databases and XML files. This data can, and then is consumed in any .NET application. ADO stands for Microsoft ActiveX Data Objects.

What are .NET Data Providers?

Databases only understand SQL. If a .NET application (Web, Windows, Console etc..) has to retrieve data, then the application needs to

1. Connect to the Database
2. Prepare an SQL Command
3. Execute the Command
4. Retrieve the results and display in the application

Different .NET Data Providers

Data Provider for SQL Server - System.Data.SqlClient
Data Provider for Oracle - System.Data.OracleClient
Data Provider for OLEDB - System.Data.OleDb
Data Provider for ODBC - System.Data.Odbc

Please note that, depending on the provider, the following ADO.NET objects have a different prefix

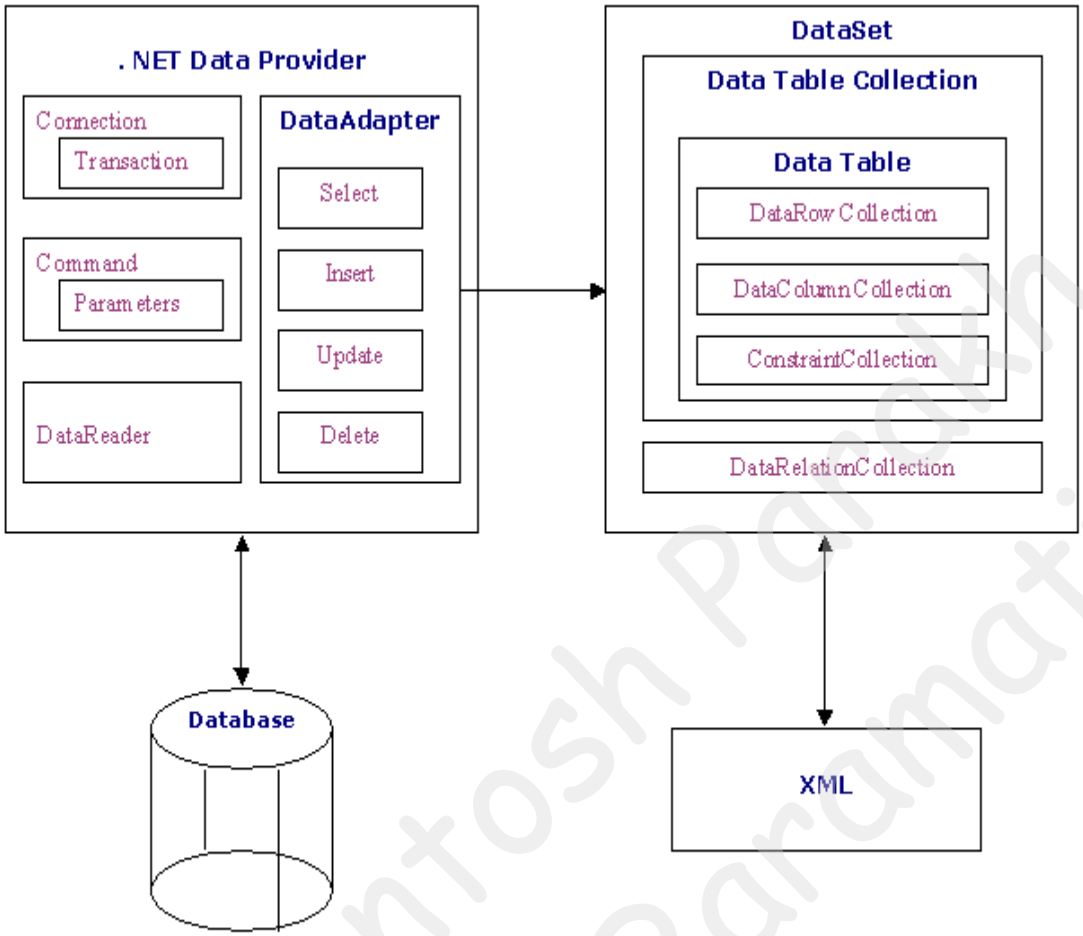
1. Connection - SQLConnection, OracleConnection, OleDbConnection, OdbcConnection etc
2. Command - SQLCommand, OracleCommand, OleDbCommand, OdbcCommand etc
3. DataReader - SQLDataReader, OracleDataReader, OleDbDataReader, OdbcDataReader etc
4. DataAdapter - SQLDataAdapter, OracleDataAdapter, OleDbDataAdapter, OdbcDataAdapter etc

The DataSet object is not provider specific. Once we connect to a Database, execute command, and retrieve data into .NET application. The data can then be stored in a DataSet and work independently of the database.

Why ADO.NET?

To cope up with some of the problems mentioned above, ADO .NET came into existence. ADO .NET addresses the above mentioned problems by maintaining a disconnected database access model which means, when an application interacts with the database, the connection is opened to serve the request of the application and is closed as soon as the request is completed. Likewise, if a database is Updated, the connection is opened long enough to complete the Update operation and is closed. By keeping connections open for only a minimum period of time, ADO .NET conserves system resources and provides maximum security for databases and also has less impact on system performance. Also, ADO .NET when interacting with the database uses XML and converts all the data into XML format for database related operations making them more efficient.

The ADO.NET Data Architecture



ADO .NET Data Architecture

Data Access in ADO.NET relies on two components: DataSet and Data Provider.

Data Provider

The Data Provider is responsible for providing and maintaining the connection to the database. A DataProvider is a set of related components that work together to provide data in an efficient and performance driven manner. The .NET Framework currently comes with two DataProviders: the SQL Data Provider which is designed only to work with Microsoft's SQL Server 7.0 or later and the OleDb DataProvider which allows us to connect to other types of databases like Access and Oracle. Each DataProvider consists of the following component classes:

The Connection object which provides a connection to the database
The Command object which is used to execute a command
The DataReader object which provides a forward-only, read only, connected recordset
The DataAdapter object which populates a disconnected DataSet with data and performs update

Data access with ADO.NET can be summarized as follows:

A connection object establishes the connection for the application with the database. The command object provides direct execution of the command to the database. If the command returns more than a single value, the command object returns a DataReader to provide the data.

Alternatively, the DataAdapter can be used to fill the Dataset object. The database can be updated using the command object or the DataAdapter.

DataSet

The dataset is a disconnected, in-memory representation of data. It can be considered as a local copy of the relevant portions of the database. The DataSet is persisted in memory and the data in it can be manipulated and updated independent of the database. When the use of this DataSet is finished, changes can be made back to the central database for updating. The data in DataSet can be loaded from any valid data source like Microsoft SQL server database, an Oracle database or from a Microsoft Access database.

Component classes that make up the Data Providers

The Connection Object

It establishes connection to a specific data source.
The base class for all connection object is **DbConnection Class**.

The Connection object creates the connection to the database. Microsoft Visual Studio .NET provides two types of Connection classes: the SqlConnection object, which is designed specifically to connect to Microsoft SQL Server 7.0 or later, and the OleDbConnection object, which can provide connections to a wide range of database types like Microsoft Access and Oracle. The Connection object contains all of the information required to open a connection to the database.

The Command Object

It executes a command against a data source
It exposes the parameters and can execute within the scope of a transaction from a connection.

The base class for all command object is **DbCommand Class**.
It requires a SQL query and connection object.

The Command object is represented by two corresponding classes: SqlCommand and OleDbCommand. Command objects are used to execute commands to a database across a data connection. The Command objects can be used to execute stored procedures on the database, SQL commands, or return complete tables directly. Command objects provide three methods that are used to execute commands on the database:
ExecuteNonQuery: Executes commands that have no return values such as INSERT, UPDATE or DELETE
ExecuteScalar: Returns a single value from a database query
ExecuteReader: Returns a result set by way of a DataReader object

The DataReader Object

It reads the data from databse.
The base class for all data reader object is **DbDataReader Class**.
It requires command object.

The DataReader object provides a forward-only, read-only, connected stream recordset from a database. Unlike other components of the Data Provider, DataReader objects cannot be directly instantiated. Rather, the DataReader is returned as the result of the Command object's ExecuteReader method. The SqlCommand.ExecuteReader method returns a SqlDataReader object,

and the OleDbCommand.ExecuteReader method returns an OleDbDataReader object. The DataReader can provide rows of data directly to application logic when you do not need to keep the data cached in memory. Because only one row is in memory at a time, the DataReader provides the lowest overhead in terms of system performance but requires the exclusive use of an open Connection object for the lifetime of the DataReader.

The DataAdapter Object

The DataAdapter is the class at the core of ADO .NET's disconnected data access. It is essentially the middleman facilitating all communication between the database and a DataSet. The DataAdapter is used either to fill a DataTable or DataSet with data from the database with it's Fill method. After the memory-resident data has been manipulated, the DataAdapter can commit the changes to the database by calling the Update method. The DataAdapter provides four properties that represent database commands:

SelectCommand
InsertCommand
DeleteCommand
UpdateCommand

When the Update method is called, changes in the DataSet are copied back to the database and the appropriate InsertCommand, DeleteCommand, or UpdateCommand is executed.

Sample ADO.NET code to connect to SQL Server Database and retrieve data. Notice that we are using **SQLConnection**, **SQLCommand** and **SQLDataReader** classes . All the objects are prefixed with the word **SQL**. All these classes are present in **System.Data.SqlClient** namespace. So, we can say that the **.NET data provider for SQL Server is System.Data.SqlClient.**

```
SqlConnection con = new SqlConnection("data source=.; database=Sample; integrated security=SSPI");
SqlCommand cmd = new SqlCommand("Select * from tblProduct", con);
con.Open();
SqlDataReader rdr = cmd.ExecuteReader();
GridView1.DataSource = rdr;
GridView1.DataBind();
con.Close();
```

Sample ADO.NET code to connect to Oracle Database and retrieve data. Notice that we are using **OracleConnection**, **OracleCommand** and **OracleDataReader** classes . All the objects are prefixed with the word **Oracle**. All these classes are present in System.Data.OracleClient namespace. So, we can say that the **.NET data provider for Oracle is System.Data.OracleClient.**

```
OracleConnection con = new OracleConnection("Oracle Database Connection String");
OracleCommand cmd = new OracleCommand("Select * from tblProduct", con);
con.Open();
OracleDataReader rdr = cmd.ExecuteReader();
GridView1.DataSource = rdr;
GridView1.DataBind();
con.Close();
```

Aadhar Card
Shop act Licen
Voter ID card
Light bill
Photo 2
Proprietor Stamp

Exception Handling

Exception handling is a builtin mechanism in .NET framework to detect and handle run time errors. The .NET framework contains many standard exceptions. The exceptions are anomalies that occur during the execution of a program. They can be because of user, logic or system errors. If a user (programmer) does not provide a mechanism to handle these anomalies, the .NET run time environment provides a default mechanism that terminates the program execution.

C# provides the three keywords try, catch and finally to do exception handling. The try block encloses the statements that might throw an exception whereas catch handles an exception if one exists. The finally can be used for doing any clean up process.

The general form of try-catch-finally in C# is shown below.

```
try
{
// Statement which can cause an exception.
}
catch(Type x)
{
// Statements for handling the exception
}
finally
{
//Any cleanup code
}
```

If any exception occurs inside the try block then the control transfers to the appropriate catch block and later to the finally block.

But in C#, both catch and finally blocks are optional. The try block can exist either with one or more catch blocks or a finally block or with both catch and finally blocks.

If there is no exception occurring inside the try block then the control directly transfers to the finally block. We can say that the statements inside the finally block is executed always. Note that it is an error to transfer control out of a finally block by using break, continue, return or goto.

In C#, exceptions are nothing but objects of the type Exception. The Exception is the ultimate base class for any exceptions in C#. The C# itself provides a couple of standard exceptions. Or even the user can create their own exception classes, provided that this should inherit from either the Exception class or one of the standard derived classes of the Exception class like DivideByZeroExcpetion or ArgumentException and so on.

Uncaught Exceptions

The following program will compile but will show an error during execution. The division by zero is a runtime anomaly and the program terminates with an error message. Any uncaught exceptions in the current context propagate to a higher context and looks for an appropriate catch block to handle it. If it can't find any suitable catch blocks then the default mechanism of the .NET

runtime will terminate the execution of the entire program.

```
//C#: Exception Handling
//Author: rajeshvs@msn.com
using System;
class MyClient
{
public static void Main()
{
int x = 0;
int div = 100/x;
Console.WriteLine(div);
}
}
```

The modified form of the preceding program with the exception handling mechanism is as follows. Here we are using the object of the standard exception class DivideByZeroException to handle the exception caused by division by zero.

```
//C#: Exception Handling
using System;
class MyClient
{
public static void Main()
{
int x = 0;
int div = 0;
try
{
div = 100/x;
Console.WriteLine("This line in not executed");
}
catch(DivideByZeroException de)
{
Console.WriteLine("Exception occured");
}
Console.WriteLine("Result is {0}",div);
}
}
```

In the preceding case the program does not terminate unexpectedly. Instead the program control passes from the point where the exception occurred inside the try block to the catch blocks. If it finds any suitable catch block then the statements inside that catch executes and continues with the normal execution of the program statements.
If a finally block is present, the code inside the finally block will also be executed.

```
//C#: Exception Handling
using System;
class MyClient
{
public static void Main()
```

```
{
int x = 0;
int div = 0;
try
{
div = 100/x;
Console.WriteLine("Not executed line");
}
catch(DivideByZeroException de)
{
Console.WriteLine("Exception occured");
}
finally
{
Console.WriteLine("Finally Block");
}
Console.WriteLine("Result is {0}",div);
}
}
```

Remember that in C#, the catch block is optional if a finally block is supplied. The following program is perfectly legal in C#.

```
//C#: Exception Handling
using System;
class MyClient
{
public static void Main()
{
int x = 0;
int div = 0;
try
{
div = 100/x;
Console.WriteLine("Not executed line");
}
finally
{
Console.WriteLine("Finally Block");
}
Console.WriteLine("Result is {0}",div);
}
}
```

But in this case, since there is no exception handling catch block, the execution will cause termination. But before the termination of the program, the statements inside the finally block will be executed. In C#, a try block must be followed by either a catch or finally block.

Multiple Catch Blocks

A try block can throw multiple exceptions that can be handled by multiple catch blocks.

Remember that a more specialized catch block should come before a generalized one. Otherwise the compiler will show a compilation error.

```
//C#: Exception Handling: Multiple catch
using System;
class MyClient
{
public static void Main()
{
int x = 0;
int div = 0;
try
{
div = 100/x;
Console.WriteLine("Not executed line");
}
catch(DivideByZeroException de)
{
Console.WriteLine("DivideByZeroException" );
}
catch(Exception ee)
{
Console.WriteLine("Exception" );
}
finally
{
Console.WriteLine("Finally Block");
}
Console.WriteLine("Result is {0}",div);
}
}
```

Catching all Exceptions

By providing a catch block without a brackets or arguments, we can catch all exceptions occurring inside a try block. We can even use a catch block with an Exception type parameter to catch all exceptions happening inside the try block, since in C# all exceptions are directly or indirectly inherited from the Exception class.

```
//C#: Exception Handling: Handling all exceptions
using System;
class MyClient
{
public static void Main()
{
int x = 0;
int div = 0;
try
{
div = 100/x;
Console.WriteLine("Not executed line");
```

```
}
catch
{
Console.WriteLine("oException" );
}
Console.WriteLine("Result is {0}",div);
}
}
```

The following program handles all exceptions with an Exception object.

```
//C#: Exception Handling: Handling all exceptions
using System;
class MyClient
{
public static void Main()
{
int x = 0;
int div = 0;
try
{
div = 100/x;
Console.WriteLine("Not executed line");
}
catch(Exception e)
{
Console.WriteLine("oException" );
}
Console.WriteLine("Result is {0}",div);
}
}
```

Throwing an Exception

In C#, it is possible to throw an exception programmatically. The "throw" statement is used for this purpose. The general form of throwing an exception is as follows.

```
throw exception_obj;
```

For example the following statement throws an ArgumentException explicitly.

```
throw new ArgumentException("Exception");
```

```
//C#: Exception Handling:
using System;
class MyClient
{
public static void Main()
{
try
{
```

```
throw new DivideByZeroException("Invalid Division");
}
catch(DivideByZeroException e)
{
Console.WriteLine("Exception" );
}
Console.WriteLine("LAST STATEMENT");
}
}
```

Re-throwing an Exception

The exceptions that we catch inside a catch block can be re-thrown to a higher context using the throw statement inside the catch block. The following program shows how to do this.

```
//C#: Exception Handling: Handling all exceptions
using System;
class MyClass
{
public void Method()
{
try
{
int x = 0;
int sum = 100/x;
}
catch(DivideByZeroException e)
{
throw;
}
}
}
class MyClient
{
public static void Main()
{
MyClass mc = new MyClass();
try
{
mc.Method();
}
catch(Exception e)
{
Console.WriteLine("Exception caught here" );
}
Console.WriteLine("LAST STATEMENT");
}
}
```

Standard Exceptions

There are two types of exceptions: exceptions generated by an executing program and exceptions generated by the common language runtime. System.Exception is the base class for all exceptions in C#. Several exception classes inherit from this class, including ApplicationException and SystemException. These two classes form the basis for most other runtime exceptions. Other exceptions that derive directly from System.Exception include IOException, WebException and so on.

The common language runtime throws SystemException. The ApplicationException is thrown by a user program rather than the runtime. The SystemException includes the ExecutionEngineException, StaclOverFlowException and so on. It is not recommended that we catch SystemExceptions nor is it good programming practice to throw SystemExceptions in our applications.

- System.OutOfMemoryException
- System.NullReferenceException
- Syste.InvalidCastException
- Syste.ArrayTypeMismatchException
- System.IndexOutOfRangeException
- System.ArithmeticException
- System.DevideByZeroException
- System.OverFlowException

User-defined Exceptions

In C#, it is possible to create our own exception class. But Exception must be the ultimate base class for all exceptions in C#. So the user-defined exception classes must inherit from either the Exception class or one of its standard derived classes.

```
//C#: Exception Handling: User defined exceptions
using System;
class MyException : Exception
{
public MyException(string str)
{
Console.WriteLine("User defined exception");
}
}
class MyClient
{
public static void Main()
{
try
{
throw new MyException("RAJESH");
}
catch(Exception e)
{
Console.WriteLine("Exception caught here" + e.ToString());
}
```

```
Console.WriteLine("LAST STATEMENT");
}
}
```

Design Guidelines

Exceptions should be used to communicate exceptional conditions. Don't use them to communicate events that are expected, such as reaching the end of a file. If there's a good predefined exception in the System namespace that describes the exception condition, one that will make sense to the users of the class, then use that one rather than defining a new exception class, and put specific information in the message. Finally, if code catches an exception that it isn't going to be handled then consider whether it should wrap that exception with additional information before re-throwing it.

State Management

What is the need of State Management?

Let us assume that someone is trying to access a banking website and he has to fill in a form. So the person fills in the form and submits it. After submission of the form, the person realizes he has made a mistake. So he goes back to the form page and he sees that the information he submitted is lost. So he again fills in the entire form and submits it again. This is quite annoying for any user. So to avoid such problems "STATE MANAGEMENT" acts as a savior for developers like us.

If you do not understand what the word "STATE" means, then think about it, as some interaction between the User and the Server.

Definition - State Management can be defined as the technique or the way by which we can maintain / store the state of the page or application until the User's Session ends.

What is HTTP Protocol?
Hyper Text Transfer Protocol is the base or I would rather say, is the core part or the foundation of our World Wide Web. It basically makes use of the TCP Protocol for communicating with the server. A specific port is being used by this protocol while communicating with the server.

If a user tries to access a website, say (http://www.asp.net/) and let us assume that the requested page is an ASPX page. So what happens is, the HTTP Protocol takes the request being sent by the client (using his browser) and sends it to the server. The server then locates the requested page and tells the ASP.NET engine to process the

request.

The ASP.NET Engine processes the request and sends a response back to the client in an HTML format. Once again the HTTP protocol is relevant; the HTTP protocol takes the response and sends it to the client, thus the response is shown in the client's browser.

So the HTTP Protocol, which is also called a "Request - Response" Protocol, acts like a bridge between the client and the server.

Some Technical Facts

As we know Web pages are based on the HTTP Protocol which is a stateless protocol, means that there is no information about the request, such as where they are coming from i.e. whether the request is from the same client or from a new client. Since the page is created each time, the page is requested then destroyed. So we can say that the state is not maintained between client requests by default.

State Management Techniques

ASP.NET provides us with 2 ways to manage the state of an application. It is basically divided into the 2 categories:

1. Client Side State Management.
2. Server Side State Management.

Client Side State Management - It is a way in which the information which is being added by the user or the information about the interaction happened between the user and the server is stored on the client's machine or in the page itself. The server resources (e.g. server's memory) are not at all utilized during the process.

This management technique basically makes use of the following:

a. View State
b. Hidden Fields
c. Query String
d. Cookies

View State

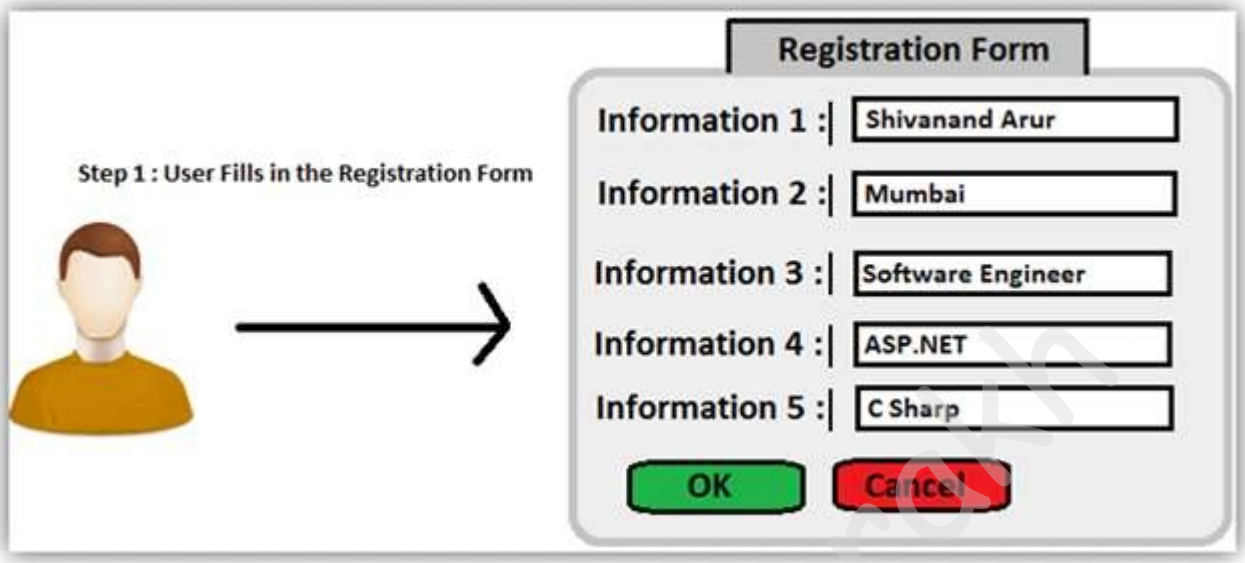
View State can be used to maintain the State at a page level. The term "Page Level" means that the information is being stored for a specific page and until that specific page is active (i.e. the page which is being currently viewed by the user). Once the user is re-directed or goes to some other page, the information stored in the View State gets lost.

It basically makes use of a "Dictionary Object" to store data, which means that the information is stored in a key and value pair. It stores this information in a Hidden field on the page itself in a hashed format. A View State can store a string value only of a specific length. If the length is exceeded then the excess information is stored in another hidden field.

Using a View State is quite simple. In fact, it is the default way for storing the page or the control's information. Typically the View State is used, when we want a user to be re-directed to the same page and the information being added by the user remains persistent until the user is on the same page.

Here I have shown how to use and assign a value to a View State and how to read a value from a View State.

Example - A user fills in a Registration Form.



Storing Value in a View State

```
protected void btnOK_Click(Object sender, EventArgs e)
{
    ViewState["Username"] = txtFName.Text;  // Storing the First Name of the User in the View State.
}
```

Information - Here in the code above, "Username" is the key and the value is the text being inputted by the user (txtFName.Text).

Retrieving Value from a View State

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!String.IsNullOrEmpty(Convert.ToString(ViewState["Username"])))
    {
        lblUsername.Text = Convert.ToString(ViewState["Username"]);
    }
}
```

Information - Here as you can see in the image, I am retrieving the value from the View State, in the Page Load event of a page by first checking if the View State is not empty or null and then assigning its value to a Label. This is just a simple example which is shown to make you understand about, how to use a View State.

View State Information is stored in a Hashed Format

```
<div>
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value="/wEPDwUKMTA1Mjg2NTQ4M2Rkutjuh48p2Tc+/aL21UtgBqbW9aY=" />
</div>
```

Information - If you look at the page source, then this is the way View State stores the value.

View State Settings

View State is customizable. With the term "Customizable" I mean, that we can apply settings to a View State to store a value at various levels. We can set the View State at various levels like:

1. **Setting View State at Application Level** - If we do not want our pages in the Application to use view state, then we can disable or enable it in the web.config file, as shown in the image below.

```
<configuration>
  <system.web>
    <pages enableViewState="false">
      <controls>
        <add tagPrefix="asp" namespace="System.Web.UI" assembly="System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
        <add tagPrefix="asp" namespace="System.Web.UI.WebControls" assembly="System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
      </controls>
    </pages>
  </system.web>
</configuration>
```

2. **Setting View State at Page Level** - If we do not want a specific page to use View State, then we can disable or enable it in the @ Page Directive which is the first line of our aspx page.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" EnableViewState="false" %>
```

3. **Setting View State at Control Level** - If we do not want a specific control to use View State, then we can disable or enable it at a Control Level as follows:

```
<td>
    <asp:TextBox ID="txtFName" runat="server" EnableViewState="false"></asp:TextBox>
</td>
```

Advantages of using a View State

1. It is very simple to use.
2. Data is stored in hashed format and hence a layman won't be able to understand the value of the View State (It still can be hacked by Hackers, so to make it more secure we should try to store the value in an encrypted format.). Check out this link http://msdn.microsoft.com/en-us/library/ms178199(v=vs.85).aspx to understand, how to secure View State.
3. It is customizable, as shown above.

Disadvantages of using a View State

1. Information is not encrypted, so it can be easy for a Hacker to get its value.
2. Cannot be used to store sensitive data (eg: Passwords, Credit Card Pins, etc).
3. Might make a page heavy if lots of data is stored in View State.

Hidden Fields

ASP.NET provides a server control called "Hidden Field" which can be used to store a value at a page level, which is similar to a View State. The value of the Hidden Field is sent along in the HTTP Form Collection along with the value of other controls.

Example - Take the same example of the User filling an online registration form.

Setting Value to a Hidden Field

```
protected void btnSubmit_Click(Object sender, EventArgs e)
{
    hdfValue.Value = txtUsername.Text;
}
```

Information - Here I am setting a value to a Hidden field, as shown in the image above. I am taking input from the User (txtUsername.Text) and assigning it to the Hidden Field's Value property. It basically stores only 1 value in its property.

Retrieving Value from a Hidden Field

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!String.IsNullOrEmpty(hdfValue.Value))
    {
        lblName.Text = hdfValue.Value;
    }
}
```

Information - As you can see, I am retrieving a value from the value in the Hidden field and assigning it to a label. The Hidden Field's "Value" property returns a string by default. If you want an integer value, then you will have to convert it explicitly.

A Hidden Field stores a value at a Page Level

If you look at the page source after assigning a value to a Hidden Field, then you will see that it stores the value on the page itself. Once the user is redirected to some other page, then the value is lost.

```
<tr>
    <td>
        <span id="lblAns"></span>

        <input type="hidden" name="hdfValue" id="hdfValue" value="Shivanand" />

    </td>
</tr>
```

I had passed "Shivanand" as a value from my textbox and assigned that value to a hidden field.

Advantages

1. Very simple to use.
2. Hidden Fields store the value in the page itself, hence do not use server resources.

Disadvantages

1. Will make a page heavy, if too many Hidden Fields are used to store data.
2. Cannot store sensitive data, as the value that is stored is neither hashed, nor encrypted.

Query String

A Query String is a string which is appended to the end of the Page URL. It is very simple to use and can be used to send data across pages. It stores information in a key - value pair. A "?" signature is used to append the key and value to the page URL.

Way to pass a value using Query String

```
protected void btnSubmit_Click(Object sender, EventArgs e)
{
    Response.Redirect(String.Format("Default.aspx?Username={0}", txtUsername.Text.Trim()));
}
```

Information - This preceding code will send the Username to another page and use that value on that page. We should never send sensitive data using Query String, since the data that is being sent can easily be tampered with by anybody. If you still want to send information using Query String, then encrypt the data using any ASP.NET Encryption technique so the data cannot be tampered with.

Way to read Query String value

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        string Name = this.Request.QueryString["Username"];
        lblUsername.Text = Name;
    }
}
```

Information - To read the value of the query string, you should use the Request Object as shown in the image above. You can send multiple parameters in the query string along with its respective value. For sending multiple parameters, you can separate the parameters using the "&"delimeter.

Cookies

ASP.Net provides another way of state management, which is by using Cookies. Cookies are one of the best ways of storing information. It is nothing but a text file which is stored on the client's machine.

When the user sends a request to the server, the server creates a cookie and attaches a header and sends it back to the user along with the response. The browser accepts the cookie and stores it at a specific location on the

client's machine. Even large sites like Gmail, Facebook, Yahoo use cookies.
There are 2 ways of assigning / storing values in cookies.

1. **Using the Request Object**

```
protected void btnSubmit_Click(Object sender, EventArgs e)
{
    this.Request.Cookies["Username"].Value = txtUsername.Text.Trim();
}
```

Information - Here I have made use of the Request object. The Cookies property of the HTTPResponse Object and the HTTPRequest Object can be used to assign values to the Cookies Collection and get values back from the Collection. We can also store multiple values in a cookie.

2. **Using the HTTPCookies Object**

```
protected void btnSubmit_Click(Object sender, EventArgs e)
{
    HttpCookie myCookie = new HttpCookie("Username",txtUsername.Text.Trim());
    myCookie["Age"] = "22";
    myCookie.Expires.AddHours(2);
    Response.Cookies.Add(myCookie);
}
```

Information - Another way of adding values to a cookie is using the "HTTPCookie" class. Its constructor takes either 1 or 2 parameters. If you want your cookie to expire after a specified time then you can even set the expiration date for that cookie. And the last line "Response.Cookies.Add(myCookie)" will add that cookie to the Cookies Collection.

Advantages

1. Very easy to use.
2. Stored on the client's machine, hence no server resources are utilized.

Disadvantages

1. A user can disable cookies using browser settings.
2. Since the cookies are stored on the client's machine, there are chances that if the client's machine is hacked then the hacker can view these cookie values. Hence we should not store sensitive information in cookies.

Server Side State Management - It is another way which ASP.NET provides to store the user's specific information or the state of the application on the server machine. It completely makes use of server resources (the server's memory) to store information.

This management technique basically makes use of the following:

a. Application State
b. Session State

Application State:

1. If the information that we want to be accessed or stored globally throughout the application, even if multiple users access the site or application at the same time, then we can use an Application Object for such purposes.
2. It stores information as a Dictionary Collection in key - value pairs. This value is accessible across the pages of the application / website.
3. There are 3 events of the Application which are as follows
    - Application_Start
    - Application_Error
    - Application_End

Example - Just for an example, I am setting the Page title in the Application Start event of the Global.asax file.

Code for setting value to the Application Object - "PageTitle" is the Key and "Welcome to State Management Application" is the value.

```
void Application_Start(object sender, EventArgs e)
{
    this.Application["PageTitle"] = "Welcome to State Management Application";
}
```

Code for reading value from the Application Object

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        this.Page.Title = Convert.ToString(this.Application["PageName"]);

    }
}
```

Session State

Session is one of the most common way which is being used by developers to maintain the state of the application. The Session basically stores the values as a dictionary collection in key/value pairs. It completely utilizes server resources to store the data. It is a secure way of storing data, since the data will never be passed to the client.

For each and every user, a separate Session is created, and each and every Session has its Unique ID. This ID is being stored in the client's machine using cookies. If there are multiple users who are accessing a web application, then for each user a separate Session is created. If a single user logs in and logs out the Session is killed, then if the same user again logs into the application, then a new Session ID is being created for the same user.

The Session has a default timeout value (20 minutes). We can also set the timeout value for a session in the web.config file.

```
<sessionState cookieless="UseUri" mode="SQLServer"></sessionState>
```

There are various ways in which we can store a session and they are as follows:

1. OFF
2. InProc
3. State Server
4. SQL Server

OFF - If we do not want to use sessions in our application, then we can set the mode as "OFF".

InProc - This is the default mode which is used in ASP.NET to store session variables. InProc mode basically stores the session value in the process in which the ASP.NET application is running. Since it is stored in the same process, it provides high performance as compared to other modes.

State Server - If we use this mode, then a separate Windows Service which runs in a different process stores the Session. It basically isolates the Session from the ASP.NET running process. It provides less performance as compared to the Inproc mode.

SQL Server - This mode stores the Session in SQL Server instead of the server's memory. If our application is stored on multiple server machines, then it becomes difficult to use the "Inproc" mode since the request can go to any server for processing. So if we want to use sessions in such cases, then we can store the Session in SQL Server so that it becomes centralized and can be accessed by any server during the request process. It is the most secure way of storing Sessions but gives the lowest performance for an application.

Code to write values into a Session

```
protected void btnSubmit_Click(Object sender, EventArgs e)
{
    Session["Username"] = txtUsername.Text.Trim();
    Response.Redirect("Default.aspx");
}
```

Code to read values from Session

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        lblUsername.Text = Convert.ToString(Session["Username"]);
    }
}
```

So this ends this article about the State Management in ASP.NET. Hope you liked it.

Web Services in ASP .Net

A Web service, in the context of .NET, is a component that resides on a Web server and provides information and services to other network applications using standard Web protocols such as HTTP and Simple Object Access Protocol (SOAP).

What is Web Service?

Web Service is an application that is designed to interact directly with other applications over the internet. In simple sense, Web Services are means for interacting with objects over the Internet. The Web Service consumers are able to invoke method calls on remote objects by using SOAP and HTTP over the Web. Web Service is language independent and Web Services communicate by using standard web protocols and data formats, such as

- HTTP
- XML
- SOAP

A Web Service is a software program that uses XML to exchange information with other software via common internet protocols.

.NET Web services provide asynchronous communications for XML applications that operate over a .NET communications framework. They exist so that users on the Internet can use applications that are not dependent on their local operating system or hardware and are generally browser-based.

A web service is

- Language Independent.
- Protocol Independent.
- Platform Independent.
- It assumes stateless service architecture.
- Scalable (e.g. multiplying two numbers together to an entire customer-relationship management system).
- Programmable (encapsulates a task).
- Based on XML (open, text-based standard).
- Self-describing (metadata for access and use).
- Discoverable (search and locate in registries)- ability of applications and developers to search for and locate desired Web services through registries. This is based on UDDI.

Web services advantages

- Use open, text-based standards, which enable components written in various languages and for different platforms to communicate.
- Promote a modular approach to programming, so multiple organizations can communicate with the same Web service.
- Comparatively easy and inexpensive to implement, because they employ an existing infrastructure and because most applications can be repackaged as Web services.

- Significantly reduce the costs of enterprise application (EAI) integration and B2B communications.
- Implemented incrementally, rather than all at once which lessens the cost and reduces the organizational disruption from an abrupt switch in technologies.
- The Web Services Interoperability Organization (WS-I) consisting of over 100 vendors promotes interoperability.

Web Services Limitations

- SOAP, WSDL, UDDI- require further development.
- Interoperability.
- Royalty fees.
- Too slow for use in high-performance situations.
- Increase traffic on networks.
- The lack of security standards for Web services.
- The standard procedure for describing the quality (i.e. levels of performance, reliability, security etc.) of particular Web services â€“ management of Web services.
- The standards that drive Web services are still in draft form (always will be in refinement).
- Some vendors want to retain their intellectual property rights to certain Web services standards.

Web Service Example

A web service can perform almost any kind of task.

- **Web Portal-** A web portal might obtain top news headlines from an associated press web service.
- **Weather Reporting-** You can use Weather Reporting web service to display weather information in your personal website.
- **Stock Quote-** You can display latest update of Share market with Stock Quote on your web site.
- **News Headline:** You can display latest news update by using News Headline Web Service in your website.
- **Email Portal**
- **SMS Portal**
- **Payment Gateway Portal**
- You can make your own web service and let others use it. For example you can make Free SMS Sending Service with footer with your companies' advertisement, so whosoever uses this service indirectly advertises your company. You can apply your ideas in N no. of ways to take advantage of it.

Frequently used word with web services

What is SOAP?

SOAP (simple object access protocol) is a remote function calls that invokes method and execute them on Remote machine and translate the object communication into XML format. In short, SOAP are way by which method calls are translate into XML format and sent via HTTP.

What is WSDL?

WSDL stands for Web Service Description Language, a standard by which a web service can tell clients what messages it accepts and which results it will return.
WSDL contains every detail regarding using web service and Method and Properties provided by web service and URLs from which those methods can be accessed and Data Types used.

What is UDDI?

UDDI allows you to find web services by connecting to a directory.

What is Discovery or .Disco Files?

Discovery files are used to group common services together on a web server. Discovery files .Disco and .VsDisco are XML based files that contains link in the form of URLs to resources that provides discovery information for a web service. Disco File contains URL for the WSDL, URL for the documentation and URL to which SOAP messages should be sent.