

Unit I - Introduction to ASP.NET

ASP.NET is a web application framework designed and developed by Microsoft. ASP.NET is open source and a subset of the [.NET Framework](#) and successor of the classic ASP(**A**ctive **S**erver **P**ages). With version 1.0 of the .NET Framework, it was first released in January 2002. So a question comes to mind that which technology we were using before the year 2002 for developing web applications and services? Answer is **Classic ASP**. So before .NET and ASP.NET there was *Classic ASP*.

ASP.NET is built on the [CLR\(Common Language Runtime\)](#) which allows the programmers to execute its code using any .NET language(C#, VB etc.). It is specially designed to work with HTTP and for web developers to create dynamic web pages, web applications, web sites, and web services as it provides a good integration of HTML, CSS, and JavaScript.

.NET Framework is used to create a variety of applications and services like Console, Web, and Windows, etc. But ASP.NET is only used to create web applications and web services. That's why we termed ASP.NET as a subset of the .NET Framework.

Below table illustrates the ASP.NET Versions History:

Year	Version
2002	1.0
2003	1.1
2005	2.0
2006	3.0
2007	3.5
2008	3.5 SP 1
2010	4.0
2012	4.5

Year	Version
2013	4.5.1
2014	4.5.2
2015	4.6
2015	4.6.1
2016	4.6.2
2017	4.7
2017	4.7.1

Note: In the year 2015, the version 5 *RC1* came and later this gets separated from the ASP.NET and turns into a new project called **ASP.NET Core Version 1.0** with some advancement.

What is Web Application?

A web application is an application installed only on the web server which is accessed by the users using a web browser like Microsoft Internet Explorer, Google Chrome, Mozilla FireFox, Apple Safari, etc. There are also some other technology like Java, PHP, Perl, Ruby on Rails, etc. which can be used to develop web applications. Web applications provide the cross-platform feature. The user needs only a web browser to access a web application. The web applications which are developed using the .NET framework or its subsets required to execute under the **Microsoft Internet Information Services(IIS)** on the server side. The work of IIS is to provide the web application's generated HTML code result to the client browser which initiates the request as shown in the below diagram.

Don't confuse in the terms ASP.NET, ASP.NET core, ASP.NET MVC, etc. ASP(Active Server Pages) supports a lot of development models which are as follows:

- **Classic ASP:** It is the first server side scripting language developed by Microsoft.

- **ASP.NET:** It is web development framework and successor of Classic ASP. ASP.NET 4.6 is the latest version.
- **ASP.NET Core:** In November 2015, Microsoft released the 5.0 version of ASP.NET which get separated later and known as ASP.NET Core. Also, it is considered as an important redesign of ASP.NET with the feature of open-source and cross-platform. Before this version, ASP.NET is only considered as Windows-only version.
- **ASP.NET Web Forms:** These are the event-driven application model which are not considered a part of the new ASP.NET Core. These are used to provide the server-side events and controls to develop a web application.
- **ASP.NET MVC:** It is the Model-View-Controller application model which can be merged with the new ASP.NET Core. It is used to build dynamic websites as it provides fast development.
- **ASP.NET Web Pages:** These are the single page application which can be merged into ASP.NET Core.
- **ASP.NET API:** It is the Web Application Programming Interface(API). Also, to create web applications ASP.NET provide the 3 development styles which are ASP.NET Web Pages, ASP.NET MVC, Web Forms.

Features of ASP.NET?

There are a lot of reasons which makes the ASP.NET popular among the developers. Some of the reasons are listed below:

Extending .NET Framework: ASP.NET is a subset of .NET Framework as it extends the .NET Framework with some libraries and tools to develop web apps. The thing that it adds to the .NET Framework is *Libraries for common web patterns like MVC, Editor Extensions, the base framework to process the web requests, and web-page templating syntax like Razor, etc.*

Performance: It is faster than the other web frameworks available in the market.

Backend Code: With the help of ASP.NET you can write the backend code for data access and any logic in [C#](#).

Dynamic Pages: In ASP.NET, Razor provides the syntax for developing the dynamic web pages with the help of C# and HTML. ASP.NET can be integrated with [JS\(JavaScript\)](#) and it also includes the frameworks like React and Angular for the SPA(Single Page Application.)

Supporting different OS: You can develop and execute ASP.NET apps on Windows, Linux, Docker, and MacOS. The Visual Studio provides the tools to build .NET apps different OS.

Web browser

A **web browser** (also referred to as an **Internet browser** or simply a **browser**) is [application software](#) for accessing the [World Wide Web](#) or a local website. When a [user](#) requests a [web page](#) from a particular [website](#), the web browser retrieves the necessary content from a [web server](#) and then displays the page on the user's device.

A web browser is not the same thing as a [search engine](#), though the two are often confused.^{[1][2]} A search engine is a website that provides [links](#) to other websites. However, to connect to a website's server and display its web pages, a user must have a web browser installed.^[3]

Web browsers are used on a range of devices, including [desktops](#), [laptops](#), [tablets](#), and [smartphones](#). In 2020, an estimated 4.9 billion people used a browser.^[4] The [most used](#) browser is [Google Chrome](#), with a 65% global market share on all devices, followed by [Safari](#) with 18%.^[5]

The most popular browsers have a number of [features](#) in common. They automatically log [browsing history](#) or can be used in a non-logging [private mode](#). They also allow users to set [bookmarks](#), customize the browser with [extensions](#), and can manage user [passwords](#).^[10] Some provide a [sync service](#) and [web accessibility](#) features.

Traditional browser arrangement: UI features above page content

Most browsers have these [user interface](#) (UI) features:

- Allow the user to open multiple [pages](#) at the same time, either in different browser windows or in different [tabs](#) of the same window.
- *Back* and *forward* buttons to go back to the previous page visited or forward to the next one.
- A *refresh* or *reload* and a *stop* button to reload and cancel loading the current page. (In most browsers, the stop button is merged with the reload button.)
- A *home* button to return to the user's [home page](#).
- An [address bar](#) to input the [URL](#) of a page and display it.
- A search bar to input terms into a [search engine](#). (In some browsers, the search bar is merged with the address bar.)

While [mobile browsers](#) have similar UI features as [desktop](#) versions, the limitations of [touchscreens](#) require mobile UIs to be simpler.^[11] The difference is significant for users accustomed to [keyboard shortcuts](#).^[12] The most popular desktop browsers also have sophisticated [web development tools](#).

Besides the common usage of graphical browsers, there are niche [text-based](#) and [headless](#) types of browsers.

Web server

A **web server** is [computer software](#) and underlying [hardware](#) that accepts requests via [HTTP](#) (the [network protocol](#) created to distribute [web content](#)) or its secure variant [HTTPS](#). A user agent, commonly a [web browser](#) or [web crawler](#), initiates communication by making a request for a [web page](#) or other [resource](#) using HTTP, and the [server](#) responds with the content of that resource or an [error message](#). A web server can also accept and store resources sent from the user agent if configured to do so.^{[1] [2]}

The hardware used to run a web server can vary according to the volume of requests that it needs to handle. At the low end of the range are [embedded systems](#), such as a [router](#) that runs a small web server as its configuration interface. A high-traffic [Internet website](#) might handle requests with hundreds of servers that run on racks of high-speed computers.

A resource sent from a web server can be a preexisting [file](#) ([static content](#)) available to the web server, or it can be generated at the time of the request ([dynamic content](#)) by another [program](#) that communicates with the server software. The former usually can be served faster and can be more easily [cached](#) for repeated requests, while the latter supports a broader range of applications.

Technologies such as [REST](#) and [SOAP](#), which use HTTP as a basis for general computer-to-computer communication, as well as support for [WebDAV](#) extensions, have extended the application of web servers well beyond their original purpose of serving human-readable pages.

HTML Form Elements

Tag	Description
<form>	Defines an HTML form for user input
<input>	Defines an input control
<textarea>	Defines a multiline input control (text area)
<label>	Defines a label for an <input> element

<code><fieldset></code>	Groups related elements in a form
<code><legend></code>	Defines a caption for a <code><fieldset></code> element
<code><select></code>	Defines a drop-down list
<code><optgroup></code>	Defines a group of related options in a drop-down list
<code><option></code>	Defines an option in a drop-down list
<code><button></code>	Defines a clickable button
<code><datalist></code>	Specifies a list of pre-defined options for input controls
<code><output></code>	Defines the result of a calculation

HTML Form Elements

This chapter describes all the different HTML form elements.

The HTML `<form>` Elements

The HTML `<form>` element can contain one or more of the following form elements:

- `<input>`
- `<label>`
- `<select>`

- `<textarea>`
- `<button>`
- `<fieldset>`
- `<legend>`
- `<datalist>`
- `<output>`
- `<option>`
- `<optgroup>`

The `<input>` Element

One of the most used form element is the `<input>` element.

The `<input>` element can be displayed in several ways, depending on the `type` attribute.

Example

```
<label for="fname">First name:</label>
<input type="text" id="fname" name="fname">
```

All the different values of the `type` attribute are covered in the next chapter: [HTML Input Types](#).

The `<label>` Element

The `<label>` element defines a label for several form elements.

The `<label>` element is useful for screen-reader users, because the screen-reader will read out loud the label when the user focus on the input element.

The `<label>` element also help users who have difficulty clicking on very small regions (such as radio buttons or checkboxes) - because when the user clicks the text within the `<label>` element, it toggles the radio button/checkbox.

The `for` attribute of the `<label>` tag should be equal to the `id` attribute of the `<input>` element to bind them together.

The <select> Element

The `<select>` element defines a drop-down list:

Example

```
<label for="cars">Choose a car:</label>
<select id="cars" name="cars">
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="fiat">Fiat</option>
  <option value="audi">Audi</option>
</select>
```

The `<option>` elements defines an option that can be selected.

By default, the first item in the drop-down list is selected.

To define a pre-selected option, add the `selected` attribute to the option:

Example

```
<option value="fiat" selected>Fiat</option>
```

Visible Values:

Use the `size` attribute to specify the number of visible values:

Example

```
<label for="cars">Choose a car:</label>
<select id="cars" name="cars" size="3">
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="fiat">Fiat</option>
  <option value="audi">Audi</option>
</select>
```

Allow Multiple Selections:

Use the `multiple` attribute to allow the user to select more than one value:

Example

```
<label for="cars">Choose a car:</label>
<select id="cars" name="cars" size="4" multiple>
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="fiat">Fiat</option>
  <option value="audi">Audi</option>
</select>
```

The <textarea> Element

The `<textarea>` element defines a multi-line input field (a text area):

Example

```
<textarea name="message" rows="10" cols="30">
The cat was playing in the garden.
</textarea>
```

The `rows` attribute specifies the visible number of lines in a text area.

The `cols` attribute specifies the visible width of a text area.



You can also define the size of the text area by using CSS:

Example

```
<textarea name="message" style="width:200px; height:600px;">
The cat was playing in the garden.
</textarea>
```

The <button> Element

The `<button>` element defines a clickable button:

Example

```
<button type="button" onclick="alert('Hello World!')">Click Me!</button>
```

This is how the HTML code above will be displayed in a browser:

Click Me!

Note: Always specify the `type` attribute for the button element. Different browsers may use different default types for the button element.

The <fieldset> and <legend> Elements

The `<fieldset>` element is used to group related data in a form.

The `<legend>` element defines a caption for the `<fieldset>` element.

Example

```
<form action="/action_page.php">
  <fieldset>
    <legend>Personalia:</legend>
    <label for="fname">First name:</label><br>
    <input type="text" id="fname" name="fname" value="John"><br>
    <label for="lname">Last name:</label><br>
    <input type="text" id="lname" name="lname" value="Doe"><br><br>
    <input type="submit" value="Submit">
  </fieldset>
</form>
```

This is how the HTML code above will be displayed in a browser:

Personalia:First name:

Last name:

The <datalist> Element

The `<datalist>` element specifies a list of pre-defined options for an `<input>` element.

Users will see a drop-down list of the pre-defined options as they input data.

The `list` attribute of the `<input>` element, must refer to the `id` attribute of the `<datalist>` element.

Example

```
<form action="/action_page.php">
  <input list="browsers">
  <datalist id="browsers">
    <option value="Internet Explorer">
    <option value="Firefox">
    <option value="Chrome">
    <option value="Opera">
    <option value="Safari">
  </datalist>
</form>
```

The <output> Element

The `<output>` element represents the result of a calculation (like one performed by a script).

Example

Perform a calculation and show the result in an `<output>` element:

```
<form action="/action_page.php"
  oninput="x.value=parseInt(a.value)+parseInt(b.value)">
```

```
0
<input type="range" id="a" name="a" value="50">
100 +
<input type="number" id="b" name="b" value="50">
=
<output name="x" for="a b"></output>
<br><br>
<input type="submit">
</form>
```

What is HTTP?

The Hypertext Transfer Protocol (HTTP) is designed to enable communications between clients and servers.

HTTP works as a request-response protocol between a client and server.

Example: A client (browser) sends an HTTP request to the server; then the server returns a response to the client. The response contains status information about the request and may also contain the requested content.

HTTP Methods

- **GET**
- **POST**

The two most common HTTP methods are: GET and POST.

The GET Method

GET is used to request data from a specified resource.

GET is one of the most common HTTP methods.

Note that the query string (name/value pairs) is sent in the URL of a GET request:

/test/demo_form.php?name1=value1&name2=value2

Some other notes on GET requests:

- GET requests can be cached
- GET requests remain in the browser history
- GET requests can be bookmarked
- GET requests should never be used when dealing with sensitive data
- GET requests have length restrictions
- GET requests are only used to request data (not modify)

The POST Method

POST is used to send data to a server to create/update a resource.

The data sent to the server with POST is stored in the request body of the HTTP request:

```
POST /test/demo_form.php HTTP/1.1
Host: w3schools.com
```

```
name1=value1&name2=value2
```

POST is one of the most common HTTP methods.

Some other notes on POST requests:

- POST requests are never cached
- POST requests do not remain in the browser history
- POST requests cannot be bookmarked
- POST requests have no restrictions on data length

Server side and Client side Programming

Server-side Programming :

It is the program that runs on server dealing with the generation of content of web page.

1) Querying the database

- 2) Operations over databases
- 3) Access/Write a file on server.
- 4) Interact with other servers.
- 5) Structure web applications.
- 6) Process user input. For example if user input is a text in search box, run a search algorithm on data stored on server and send the results.

Examples :

The Programming languages for server-side programming are :

- 1) PHP
- 2) C++
- 3) Java and JSP
- 4) Python
- 5) Ruby on Rails

Refer [PHP articles](#) for example server side codes.

Client-side Programming :

It is the program that runs on the client machine (browser) and deals with the user interface/display and any other processing that can happen on client machine like reading/writing cookies.

- 1) Interact with temporary storage
- 2) Make interactive web pages
- 3) Interact with local storage
- 4) Sending request for data to server
- 5) Send request to server
- 6) work as an interface between server and user

The Programming languages for client-side programming are :

- 1) Javascript
- 2) VBScript
- 3) HTML
- 4) CSS
- 5) AJAX

Difference Between Server-side Scripting and Client-side Scripting

Comparison Chart

BASIS FOR COMPARISON	SERVER-SIDE SCRIPTING	CLIENT-SIDE SCRIPTING
Basic	Works in the back end which could not be visible at the client end.	Works at the front end and script are visible among the users.
Processing	Requires server interaction.	Does not need interaction with the server.
Languages involved	PHP, ASP.net, Ruby on Rails, ColdFusion, Python, etcetera.	HTML, CSS, JavaScript, etc.
Affect	Could effectively customize the web pages and provide dynamic websites.	Can reduce the load to the server.
Security	Relatively secure.	Insecure

Web Form Life Cycle

Every request for a page made from a web server causes a chain of events at the server. These events, from beginning to end, constitute the life cycle of the page and all its components. The life cycle begins with a request for the page, which causes the server to load it. When the request is complete, the page is unloaded. From one end of the life cycle to the other, the goal is to render appropriate HTML output back to the requesting browser. The life cycle of a page is marked by the following events, each of which you can handle yourself or leave to default handling by the ASP.NET server:

Initialize: Initialize is the first phase in the life cycle for any page or control. It is here that any settings needed for the duration of the incoming request are initialized.

Load ViewState: The ViewState property of the control is populated. The ViewState information comes from a hidden variable on the control, used to persist the state across round trips to the server. The input string from this hidden variable is parsed by the page framework, and the ViewState property is set. This can be modified via the LoadViewState() method: This allows ASP.NET to manage the state of your control across page loads so that each control is not reset to its default state each time the page is posted.

Process Postback Data: During this phase, the data sent to the server in the posting is processed. If any of this data results in a requirement to update the ViewState, that update is performed via the LoadPostData () method.

Load: CreateChildControls () is called, if necessary, to create and initialize server controls in the control tree. State is restored, and the form controls show client-side data. You can modify the load phase by handling the Load event with the OnLoad method.

Send Postback Change Modifications: If there are any state changes between the current state and the previous state, change events are raised via the RaisePostDataChangedEvent () method.

Handle Postback Events: The client-side event that caused the postback is handled.

PreRender: This is the phase just before the output is rendered to the browser. It is essentially your last chance to modify the output prior to rendering using the `OnPreRender ()` method.

Save State: Near the beginning of the life cycle, the persisted view state was loaded from the hidden variable. Now it is saved back to the hidden variable, persisting as a string object that will complete the round trip to the client. You can override this using the `SaveViewState ()` method.

Render: This is where the output to be sent back to the client browser is generated. You can override it using the `Render` method. `CreateChildControls ()` is called, if necessary, to create and initialize server controls in the control tree.

Dispose: This is the last phase of the life cycle. It gives you an opportunity to do any final cleanup and release references to any expensive resources, such as database connections. You can modify it using the `Dispose ()` method.

Page Event

Page Event	Typical Use
PreInit	This event is raised after the start stage is complete and before the initialization stage.
Init	This event occurs after all controls have been initialized. We can use this event to read or initialize control properties.
InitComplete	This event occurs at the end of the page's initialization stage. We can use this event to make changes to view state that we want to make sure are persisted after the next postback.

PreLoad	This event is occurs before the post back data is loaded in the controls.
Load	This event is raised for the page first time and then recursively for all child controls.
Control events	This event is used to handle specific control events such as Button control' Click event.
LoadComplete	This event occurs at the end of the event-handling stage.We can use this event for tasks that require all other controls on the page be loaded.
PreRender	This event occurs after the page object has created all controls that are required in order to render the page.
PreRenderComplete	This event occurs after each data bound control whose DataSourceID property is set calls its DataBind method.
SaveStateComplete	It is raised after view state and control state have been saved for the page and for all controls.
Render	This is not an event; instead, at this stage of processing, the Page object calls this method on each control.
Unload	This event raised for each control and then for the page.

HTTP Request Response Structure

Whether you are a user or a website owner, the one word you might come across when browsing is HTTP. It is important to get the basics of HTTP to understand how Internet works and the details sent and received between your browser and the web server.

What is HTTP?

HTTP stands for **H**yper**T**ext **T**ransfer **P**rotocol. This is a basis for data communication in the internet. The data communication starts with a request sent from a client and ends with the response received from a web server.

- A website URL starting with “http://” is entered in a web browser from a computer (client). The browser can be a Chrome, Firefox, Edge, Safari, Opera or anything else.
 - Browser sends a request sent to the web server that hosts the website.
 - The web server then returns a response as a HTML page or any other document format to the browser.
 - Browser displays the response from the server to the user.
- The symbolic representation of a HTTP communication process is shown in the below picture:

The web browser is called as a User Agent and other example of a user agent is the crawlers of search engines like Googlebot.

1 HTTP Request Structure from Client

A simple request message from a client computer consists of the following components:

- A request line to get a required resource, for example a request GET /content/page1.html is requesting a resource called /content/page1.html from the server.
 - Headers (Example – Accept-Language: EN).
 - An empty line.
 - A message body which is optional.
- All the lines should end with a carriage return and line feed. The empty line should only contains carriage return and line feed without any spaces.

2 HTTP Response Structure from Web Server

A simple response from the server contains the following components:

- HTTP Status Code (For example HTTP/1.1 301 Moved Permanently, means the requested resource was permanently moved and redirecting to some other resource).
- Headers (Example – Content-Type: html)
- An empty line.
- A message body which is optional.

All the lines in the server response should end with a carriage return and line feed. Similar to request, the empty line in a response also should only have carriage return and line feed without any spaces.

3. Example of HTTP Session

Let us take an example that you want to open a page “home.html” from the site “yoursite.com”. Below is how the request from the client browser should look like to get a “home.html” page from “yoursite.com”.

```
GET /home.html HTTP/1.1

Host: www.yoursite.com
```

The response from the web server should look like below:

```
HTTP/1.1 200 OK

Date: Sun, 28 Jul 2013 15:37:37 GMT

Server: Apache

Last-Modified: Sun, 07 Jul 2013 06:13:43 GMT

Transfer-Encoding: chunked

Connection: Keep-Alive

Content-Type: text/html; charset=UTF-8

Webpage Content
```

Chunked transfer encoding is a method in which the server responds with a data in chunks and this used in place of Content-Length header. The communication is stopped when a zero length of chunk is received and this method is used in HTTP Version 1.1.

4. What is that HTTPS?

Now you understand HTTP then what is that HTTPS? HTTPS is the secured HTTP protocol required to send and receive information securely over internet. Nowadays it is mandatory for all websites to have HTTPS protocol to have secured internet. Browsers like Google Chrome will show an alert with “Not Secure” message in the address bar if the site is not served over HTTPS.

Besides the security and encryption, the communication structure of HTTPS protocol remains same as HTTP protocol as explained above.

Warning: We do not recommend using confidential information like credit card details on HTTP sites. Ensure the financial transactions happens through HTTPS protocol.

5. How to Check HTTP Request and Response on Chrome?

Well, it's time to practical. Let us take Google Chrome the popular browser, but the process remains same in all other browsers to view the details.

- Open a webpage in Google Chrome and go to “View > Developer > Developer Tools” menu.
- You can also open the developer console by right clicking on the page and choose “Inspect” option.
- Go to “Network” tab and then reload the page. Now you will see the loading time for each single component on the page.
- Click on the “Show Overview” icon to remove the timeline so that you can view other details clearly.
- Click the page URL on the left bar and go to “Response” tab. (You can also view the same details under “Preview” tab).

Viewing HTTP Request and Response in Google Chrome

You can see the details of request and responses as exactly we have explained in the above sections. The “Headers” tab will show you the details of HTTP header information for request and response for the selected item.

6. HTTP Header Checker Tool

Similar to Chrome, there are also many other free tools available to check the response code received in HTTP headers. For example, go to this [HTTP header checker tool](#), enter any of the URL you wanted to check and click the submit button.

validation controls?

Validation is important part of any web application. User's input must always be validated before sending across different layers of the application.¹

Validation controls are used to,

- Implement presentation logic.
- To validate user input data.
- Data format, data type and data range is used for validation.

Validation is of two types

1. Client Side
2. Server Side

Client side validation is good but we have to be dependent on browser and scripting language support.

Client side validation is considered convenient for users as they get instant feedback. The main advantage is that it prevents a page from being postback to the server until the client validation is executed successfully.

For developer point of view server side is preferable because it will not fail, it is not dependent on browser and scripting language.

You can use ASP.NET validation, which will ensure client, and server validation. It work on both end; first it will work on client validation and then on server validation. At any cost server validation will work always whether client validation is executed or not. So you have a safety of validation check.

For client script .NET used JavaScript. WebUIValidation.js file is used for client validation by .NET

Validation Controls in ASP.NET

An important aspect of creating ASP.NET Web pages for user input is to be able to check that the information users enter is valid. ASP.NET provides a set of validation controls that provide an easy-to-use but powerful way to check for errors and, if necessary, display messages to the user.

There are six types of validation controls in ASP.NET

1. RequiredFieldValidation Control
2. CompareValidator Control
3. RangeValidator Control
4. RegularExpressionValidator Control
5. CustomValidator Control
6. ValidationSummary

The below table describes the controls and their work,

Validation Control	Description
RequiredFieldValidation	Makes an input control a required field
CompareValidator	Compares the value of one input control to the value of another input control or to a fixed value
RangeValidator	Checks that the user enters a value that falls between two values
RegularExpressionValidator	Ensures that the value of an input control matches a specified pattern
CustomValidator	Allows you to write a method to handle the validation of the value entered
ValidationSummary	Displays a report of all validation errors occurred in a Web page

All validation controls are rendered in form as (label are referred as on client by server)

Important points for validation controls

- ControlToValidate property is mandatory to all validate controls.
- One validation control will validate only one input control but multiple validate control can be assigned to a input control.

Different Navigation Controls in ASP.NET

There are three navigation control in ASP.NET:

- SiteMapPath
- Menu Control
- TreeView

1. SiteMapPath Control

Site maps are XML files which are mainly used to describe the logical structure of the web application. It defines the layout of all pages in web application and how they relate to each other. Whenever you want you can add or remove pages to your site map there by managing navigation of website efficiently. Site map files are defined with .sitemap extension. <sitemap> element is the root node of the sitemap file.

It has three attributes:

- **Title:** It provides textual description of the link.
- **URL:** It provides the location of the valid physical file.
- **Description:** It is used for tooltip of the link.

SiteMapPath control displays the navigation path of the current page.

The path acts as click able links to previous page. The sitemap path control uses the web. This control creates the navigation mechanism which is linear path defining where the user is currently located in navigation arrangement. It helps end user to know his location in relation to the rest of the site.

Properties of SiteMapPath Control:

- **PathSeparator:** This property is to get or set the out separator text.
- **NodeStyle:** This property is used to set the style of all nodes that will be displayed.
- **RootNodeStyle:** This property is used to set the style on the absolute root node.

- **PathDirection:** This property is used to set the direction of the links generated in the output.
- **CurrentNodeStyle:** This property is used to set the style on node that represent the current page.
- **ShowToolTips:** This property is used to set the tooltip for the control. Default value is true.
- **PathSeparatorStyle:** This property is used to set the style of path separator.

2. Menu Control

Menu is another navigation control in ASP.NET which is used to display menu in web page. This control is used in combination with SiteMapDataSource control for navigating the web Site. It displays two types of menu static menu and dynamic menu. Static menu is always displayed in menu Control, by default only menu items at the root levels are displayed. Dynamic menu is displayed only when the user moves the mouse pointer over the parent menu that contains a dynamic sub menu.

Properties of Menu Control:

- **DataSourceID:** This property is used to specify the data source to be used using sitemap file as data source.
- **CssClass:** This property is used to specify the CSS class attribute for the control.
- **ImageUrl:** This property is used to specify the image that appear next to the menu item.
- **Orientation:** This property is used to specify the alignment of menu control. It can be horizontal or vertical.
- **Tooltip:** This property is used to specify the tooltip of the menu item when you mouse over.
- **Text:** This property is used to specify the text to display in the menu.
- **NavigateUrl:** This property is used to specify the target location to send the user when menu item is clicked.
- **Target:** This property is used to specify the target page location. It can be in new window or same window.
- **Value:** This property is used to specify the unique id to use in server side events.

3. *TreeView Control*

TreeView is another navigation control used in ASP.NET to display the data in hierarchical list manner. When TreeView is displayed for the first time, it displays all of its nodes. User can control it by setting the property called ExpandDepth.

Properties of TreeView Control:

- **DataSourceID:** This property is used to specify the data source to be used using sitemap file's data source.
- **ShowLines:** This property is used to specify the lines to connect the individual item in the tree.
- **CssClass:** This property is used to specify the CSS class attribute for the control.
- **ExpandDepth:** This property is used to specify the level at which items in the tree are expanded.

ASP.NET Web Forms - Master Pages

[« Previous](#)

[Next Chapter »](#)

Master pages provide templates for other pages on your web site.

Master Pages

Master pages allow you to create a consistent look and behavior for all the pages (or group of pages) in your web application.

A master page provides a template for other pages, with shared layout and functionality. The master page defines placeholders for the content, which can be overridden by content pages. The output result is a combination of the master page and the content page.

The content pages contain the content you want to display.

When users request the content page, ASP.NET merges the pages to produce output that combines the layout of the master page with the content of the content page.

Master Page Example

```
<%@ Master %>
```

```
<html>
<body>
<h1>Standard Header From Masterpage</h1>
<asp:ContentPlaceholder id="CPH1" runat="server">
</asp:ContentPlaceholder>
</body>
</html>
```

The master page above is a normal HTML page designed as a template for other pages.

The **@ Master** directive defines it as a master page.

The master page contains a placeholder tag **<asp:ContentPlaceholder>** for individual content.

The **id="CPH1"** attribute identifies the placeholder, allowing many placeholders in the same master page.

This master page was saved with the name "**master1.master**".

Content Page Example

```
<%@ Page MasterPageFile="master1.master" %>

<asp:Content ContentPlaceHolderId="CPH1" runat="server">
    <h2>Individual Content</h2>
    <p>Paragraph 1</p>
    <p>Paragraph 2</p>
</asp:Content>
```

The content page above is one of the individual content pages of the web.

The **@ Page** directive defines it as a standard content page.

The content page contains a content tag **<asp:Content>** with a reference to the master page (ContentPlaceHolderId="CPH1").

This content page was saved with the name "**mypage1.aspx**".

When the user requests this page, ASP.NET merges the content page with the master page.

ASP.NET UNIT-2

Cross Page Posting In ASP.NET

ASP.NET by default, submits the form to the same page. Cross page posting is submitting the form to a different page. This is usually required when you are creating a multi page form to collect information from the user on each page. When moving from the source to the target page, the values of controls in the source page can be accessed in the target page.

To use cross-page posting, you have to use the `PostBackUrl` attribute to specify the page you want to post to.

Follow these steps :

Step 1: Create a new ASP.NET website called `CrossPagePosting`. By default, the website is created with a single webpage, `Default.aspx`. Right click the project in the Solution Explorer > Add New Item > Web Form. Keep the original name `Default2.aspx` and click 'Add'. The website will now contain two pages, `Default.aspx` and `Default2.aspx`.

Step 2: On the source page, `Default.aspx`, drop a button on the form. Set the text of the button as 'TargetButton'. Set the 'PostBackUrl' property of a Button to the URL of the target page, `Default2.aspx`.

```
<asp:Button ID="Button1" runat="server" PostBackUrl="~/Default2.aspx" Text="TargetButton" /></div>
```

Step 3: In the target page `Default2.aspx`, drop a label on the page from the toolbox.

Step 4: In the `Page_Load()` of `Default2.aspx`, you can then access the 'PreviousPage' property to check if the page is being accessed as Cross Page postback.

```
protected void Page_Load(object sender, EventArgs e)
{
    if (Page.PreviousPage != null)
    {
    }
}
```

Step 5: To retrieve values from the source page, you must access controls using the 'FindControl()' method of the 'PreviousPage'. We will be accessing the Text property of the Button control placed in `Default.aspx`.

```
protected void Page_Load(object sender, EventArgs e)
{
    if (Page.PreviousPage != null)
    {
        Button btn =
        (Button) (Page.PreviousPage.FindControl("button1"));
    }
}
```

```
        Label1.Text = btn.Text;
    }
}
```

Step 6: In the Solution Explorer, right click Default.aspx > 'Set as Start Page'. Run the application and click on the button. As you can observe, the page is posted to Default2.aspx and the value containing the name of the button control gets displayed in the label.

Response.Redirect

When you use `Response.Redirect("Default.aspx", false)` then the execution of current page is not terminated instead of this code written after the `Response.Redirect` is executed and then after that page is redirected to the given page.

When you use `Response.Redirect("Default.aspx", true)` which is by default `true` then the execution of current page is terminated and code written after the `Response.Redirect` is not executed instead of executing code written after the `Response.Redirect`, the page is redirected to the given page.

1. `Response.Redirect` is the method of Aps.net which is used to transfer the page from one page to another.
2. `Response.Redirect` method takes two parameter URL and `endResponse`.
3. `Response.Redirect` has URL is the mandatory parameter where as `endResponse` is optional parameter.
4. url is the string parameter which takes the url or page name and `endResponse` is the boolean parameter which has true and false values.
5. By default `Response.Redirect` has `EndResponse` value is true.
6. `Response.Redirect("Default.aspx", false)` means current page execution is not terminated and code written after the `Response.Redirect("Default.aspx", false)` is executed and then after the page is redirected to the Default.aspx.
7. `Response.Redirect("Default.aspx", true)` means current page execution is terminated and page is redirected to the default.aspx page without executing the code which is written after the `Response.Redirect("Default.aspx", true)`

Difference between Server.Transfer and Cross Page Posting

Response.Redirect

1. `Response.Redirect()` will send you to a new page, update the address bar and add it to the Browser History. On your browser you can click back.

2. It redirects the request to some plain HTML pages on our server or to some other web server.
3. It causes additional roundtrips to the server on each request.
4. It doesn't preserve Query String and Form Variables from the original request.
5. It enables to see the new redirected URL where it is redirected in the browser (and be able to bookmark it if it's necessary).
6. Response. Redirect simply sends a message down to the (HTTP 302) browser.

Server.Transfer

1. Server.Transfer() does not change the address bar, we cannot hit back. One should use Server.Transfer() when he/she doesn't want the user to see where he is going. Sometime on a "loading" type page.
2. It transfers current page request to another .aspx page on the same server.
3. It preserves server resources and avoids the unnecessary roundtrips to the server.
4. It preserves Query String and Form Variables (optionally).
5. It doesn't show the real URL where it redirects the request in the users Web Browser.
6. Server.Transfer happens without the browser knowing anything, the browser request a page, but the server returns the content of another.

What is HiddenField?

HiddenField, as name implies, is hidden. This is non visual control in ASP.NET where you can save the value. This is one of the types of client-side state management tools. It stores the value between the roundtrip. Anyone can see HiddenField details by simply viewing the source of document.

HiddenFields are not encrypted or protected and can be changed by anyone. However, from a security point of view, this is not suggested. ASP.NET uses HiddenField control for managing the ViewState. So, don't store any important or confidential data like password and credit card details with this control.

```
1. <asp:HiddenFieldID="HiddenField1" runat="server" />
```

Use of HiddenField

We developers mostly do not show an ID value of table like ProductID, MemberID because users are not concerned with this kind of data. We store that information in HiddenFields and complete our process very easily.

Events of HiddenFields

As a control, it should have events. HiddenFields has the following events.

EVENT TYPE	DESCRIPTION
DataBinding	Occurs when Server Control binds to a data source.
Disposed	Occurs when Server Control is released from the memory.
Init	Occurs when Server Control is initialized.
Load	Occurs when server control get loaded on the page.
PreRender	Occurs before Rendering the page.
UnLoad	Occurs when Server Control is unloaded from memory.
ValueChanged	Occurs when the value gets changed between the round-trip (postback).

ValueChanged event is server-side control. This event gets executed when the value of HiddenField gets changed between postback to the Server.

Nowadays, people avoid using server side ValueChanged Event because all these things are possible through JavaScript or jQuery very easily.

Store the value in HiddenField

```
1. <asp:HiddenFieldID="hdnflldCurrentDateTime" runat="server" />
```

Set the value of HiddenField in code behind

```
1. protectedvoid Page_Load(object sender, EventArgs e) {  
2.     hdnflldCurrentDateTime.Value = DateTime.Now.ToString();  
3. }
```

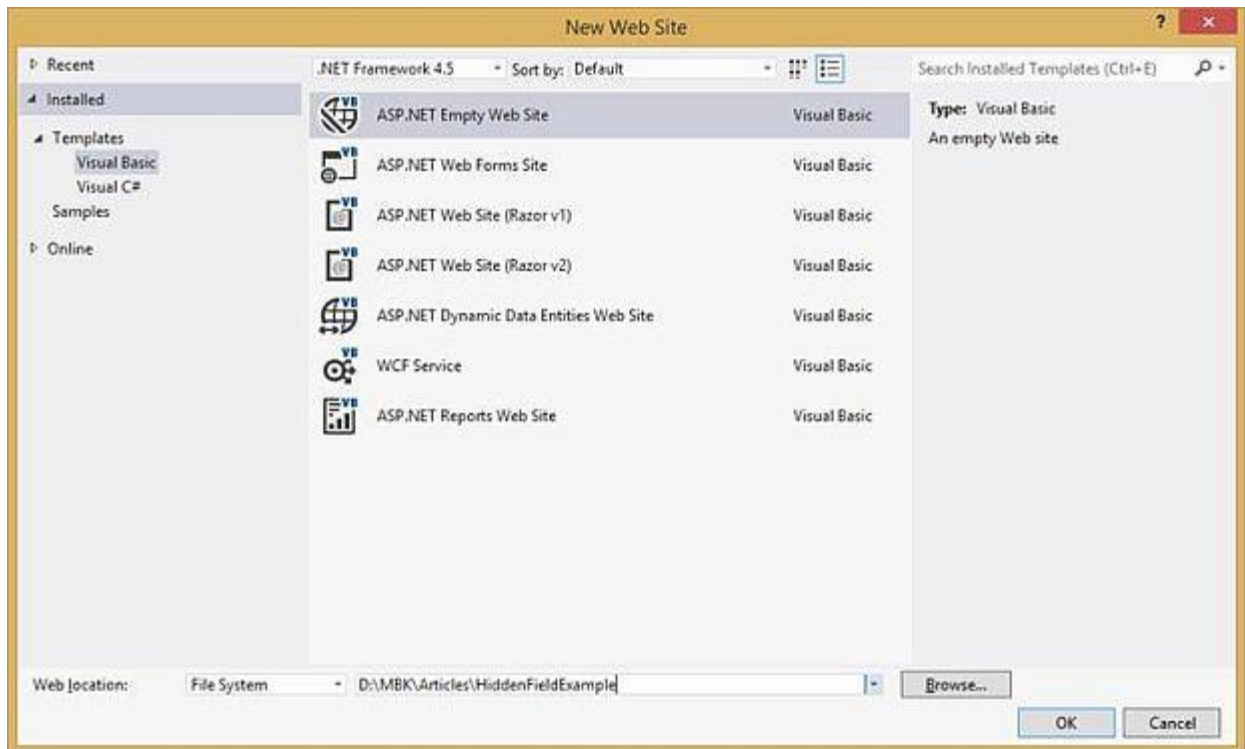
You can visit this [article](#) to see the right uses of HiddenField.

Retrieve the value from HiddenField

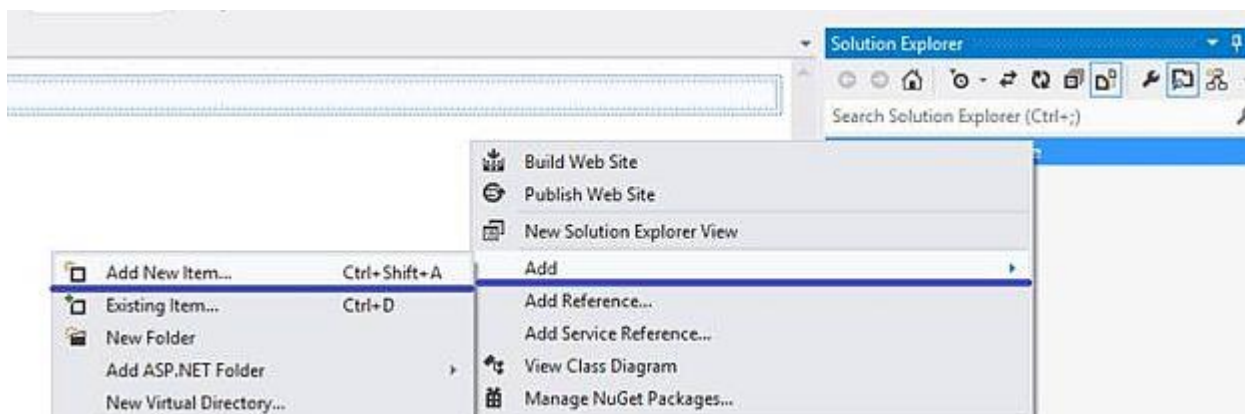
```
1. lblCurrentDateTime.Text = Convert.ToString(hdnflldCurrentDateTime  
    .Value);
```

Step by step implementation

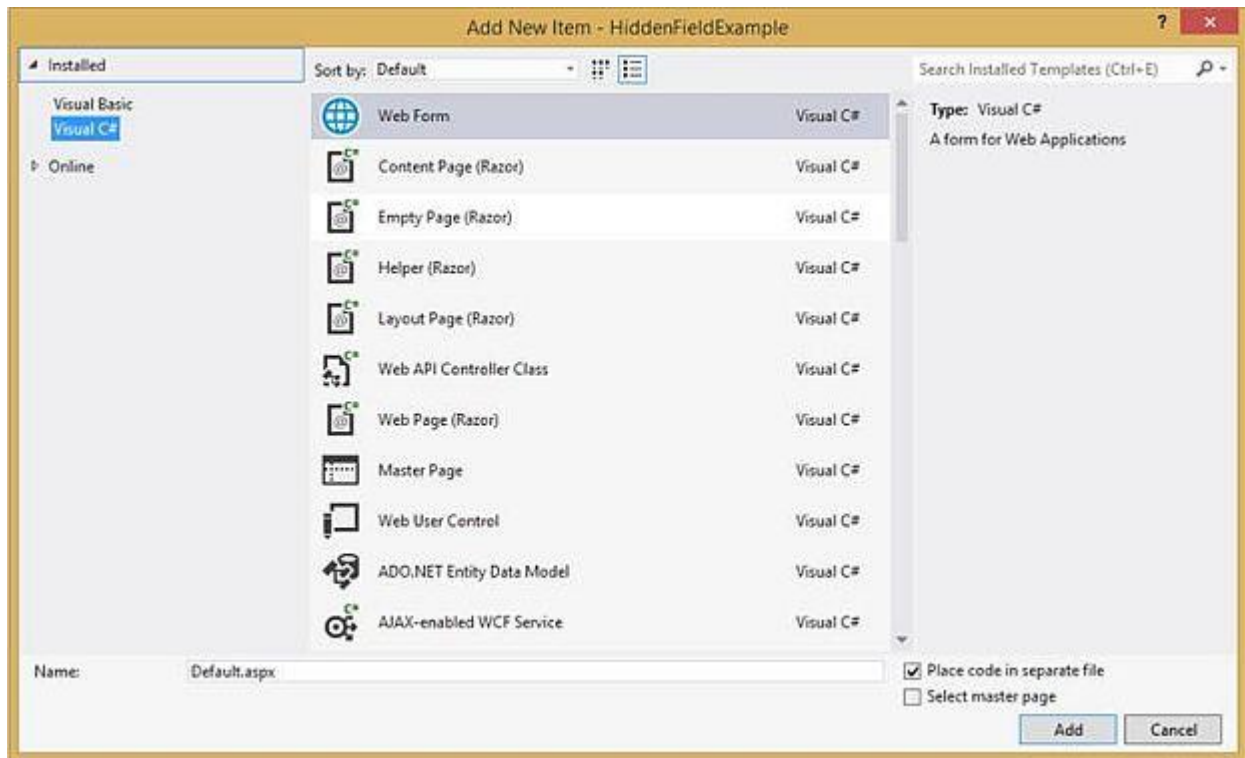
Create a new ASP.NET Website project called “HiddenFieldExample”.



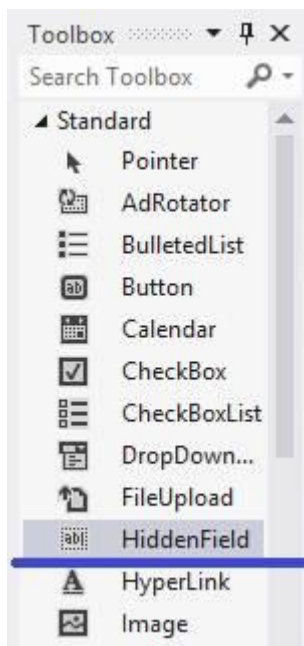
Right click on project and select Add-->Add New Item and select Web Form.



Add a new Web Form called "Default.aspx".



Now, drag and drop the HiddenField Control on the page.



By default, the HiddenField control looks like this.

```
1. <asp:HiddenFieldID="HiddenField1" runat="server" />
```

Default.aspx Code

```

1. <%@PageLanguage="C#"AutoEventWireup="true"CodeFile="Default.aspx
   .cs"Inherits="_Default"%>
2.     <!DOCTYPEhtml>
3.     <htmlxmlns="http://www.w3.org/1999/xhtml">
4.         <headrunat="server">
5.             <title></title>
6.         </head>
7.
8.         <body>
9.             <formid="form1" runat="server">
10.                 <div>
11.                     <asp:HiddenFieldID="hdnflldCurrentDateT
   ime" runat="server" />
12.                     <asp:LabelID="lblCurrentDateTime" runa
   t="server" Text=""></asp:Label>
13.                 </div>
14.             </form>
15.         </body>
16.
17.     </html>

```

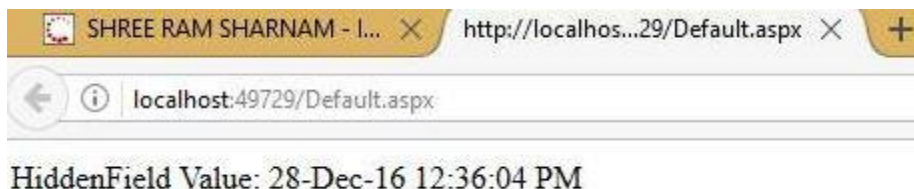
Default.aspx.cs Code

```

1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Web;
5. using System.Web.UI;
6. using System.Web.UI.WebControls;
7. public partial class_Default: System.Web.UI.Page {
8.     protected void Page_Load(object sender, EventArgs e) {
9.         hdnflldCurrentDateTime.Value = DateTime.Now.ToString();
10.         lblCurrentDateTime.Text = Convert.ToString(hdnflldCurre
   ntDateTime.Value);
11.     }
12. }

```

Output



View State

View State is a technique to maintain the state of controls during page post-back, meaning it stores the page value at the time of post-back (sending and receiving information from the server) of your page and the view state data can be used when the page is posted back to the server and a new instance of the page is created.

View state data is nothing but a serialized base-64 encoded string stored in a hidden input field on the page and it travels between the browser and the server on every user request and response.

- ```
1. ViewState["VarName"] = store any thing
```

On the web server it is encoded on the page init event of the page's life and assigned back to the variable and after the HTML rendering and sent back to the browser with the values to client.

- ```
1. <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value="
mlqif/yufT121LcPxuR5TVSuWVDJ7aU+2ONZy5gYWjTgmggCv5ed40lAOS+jpYlW
SI1hLbIA0cyrLI2Y0ZPo4RIESahtyWmLMhXbfEJ/GvJIvbfEE+JShtDaw2iFc/km
z73T0oifsuZN6JzufE1ZI+NL7qrjzp0c9PTadu+Qxxokyw7cfV6ISa+fu9qSmjpY
sxVtyxg/Z0QTyZBRaUiMbxWEJNlH3csR1d8HCPtoZ2s=" />
```

```
<!DOCTYPE html>
<html lang="en">
<head><meta charset="utf-8" /><title>
    Home Page - My ASP.NET Application
</title><script src="/Scripts/modernizr-2.6.2.js"></script>
<link href="/Content/Site.css" rel="stylesheet"/>
<link href="favicon.ico" rel="shortcut icon" type="image/x-icon" /><meta name="viewport" content="width=device-width" /></head>
<body>
    <form method="post" action="Default.aspx" id="ctl01">
<div class="aspNetHidden">
<input type="hidden" name="__EVENTTARGET" id="__EVENTTARGET" value="" />
<input type="hidden" name="__EVENTARGUMENT" id="__EVENTARGUMENT" value="" />
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value="elqif/yufl12LlCpXuR5TVSuhVDJ7aU+
20NIz5gWjTgggcV5ed40lA0S+jpYLH5IihlBIA0cyrlI2YOZPo4RIESahtyWmLPmXbF7/GvJ3IvbfEE+JSHdDaw21Fc/kmz73T0oifsuZW6JzuFEIIZl7nR7qzpc09P7adu+Qxxokyw7cfV6ISa+fu9q5mjpySxV
lH3csR1d0HCpCtoZ2s=" />
</div>

<script type="text/javascript">
//
var theForm = document.forms['ctl01'];
if (!theForm) {
    theForm = document.ctl01;
</pre>
```

Sample Code:

```
1. namespace Test
2. {
3.     public partial class _Default : Page
4.     {
5.         int TestCounter = 0;
6.         protected void Page_Load(object sender, EventArgs e)
7.         {
8.             if(!IsPostBack)
9.             {
10.                 TextBox1.Text = "0";
11.                 TextBox2.Text = "0";
12.             }
13.         }
14.     }
15. }
```

```

13.         }
14.         protected void Button1_Click(object sender, EventArgs
15.         e)
16.         {
17.             //With out View State
18.             TestCounter = TestCounter + 1;
19.             TextBox1.Text = TestCounter.ToString();
20.             if (ViewState["counter"] != null)
21.             {
22.                 TestCounter = (int)ViewState["counter"] + 1;
23.                 TextBox2.Text = TestCounter.ToString();
24.             }
25.             ViewState["counter"] = TestCounter;
26.
27.         }
28.     }
29. }

```

View State advantages:

- Very easy to implement.
- Stored on the client browser in a hidden field as a form of Base64 Encoding String not encrypted and can be decoded easily.
- Good with HTTP data transfer

View State disadvantages:

- The performance overhead for the page is larger data stored in the view state.
- Stored as encoded and not very safe to use with sensitive information.

Where to Use

View state should be used when the user needs to store a small amount of data at the client browser with faster retrieval. The developer should not use this technique to retain state with larger data since it will create a performance overhead for the webpage. It should be used for sending data from one page to another. Not very secure to store sensitive information.

ASP.NET Session State

Session State is another state management technique to store state, meaning it helps in storing and using values from previous requests. Whenever the user requests a web form from a web application it will get treated as a new request. an ASP.NET session will be used to store the previous requests for a specified time period.

```

1. //Stored Textbox value
2. Session["Counter"] = TextBox3.Text;
3.
4. //Stored Dataset
5.
6. Session["ds"] = dsData;

```

7. Session variables are stored **in** a SessionStateItemCollection **object** that **is** exposed through the HttpContext.Session property.

Session variables are stored in a SessionStateItemCollection object that is exposed through the HttpContext.Session property.

By default, an ASP.NET session state is enabled for all ASP.NET applications. Session state data is shared across all the web pages, in other words when navigating from one page session the data would available.

```
1. namespace Test
2. {
3.     public partial class SessionTest : System.Web.UI.Page
4.     {
5.         protected void Page_Load(object sender, EventArgs e)
6.         {
7.             if (!IsPostBack)
8.             {
9.                 if (Session["Counter"] == null)
10.                {
11.                    Session["Counter"] = 0;
12.                }
13.                TextBox1.Text = Session["Counter"].ToString();
14.
15.            }
16.        }
17.        protected void Button1_Click(object sender, EventArgs
18.        e)
19.        {
20.            if (Session["Counter"] != null)
21.            {
22.                int SessionCounter = (int)Session["Counter"] +
23.                1;
24.                TextBox1.Text = SessionCounter.ToString();
25.                Session["Counter"] = SessionCounter;
26.            }
27.        }
28.        protected void Button2_Click(object sender, EventArgs
29.        e)
30.        {
31.            Response.Redirect("MySessionPage.aspx");
32.        }
33.    }
34. After navigating to the page mysessionpage.aspx and retrieving
    value from session.
```



```

35.
36. namespace Test
37. {
38.     public partial class MySessionPage : System.Web.UI.Page
39.     {
40.         protected void Page_Load(object sender, EventArgs e)
41.         {
42.             if (Session["Counter"] != null)
43.             {
44.                 Label1.Text = Session["Counter"].ToString();
45.             }
46.         }
47.     }
48. }

```

Session variables are single-user global data stored on the web server, meaning by default a session state variable is stored in the web server memory and is available across all pages but it will be for a single session.

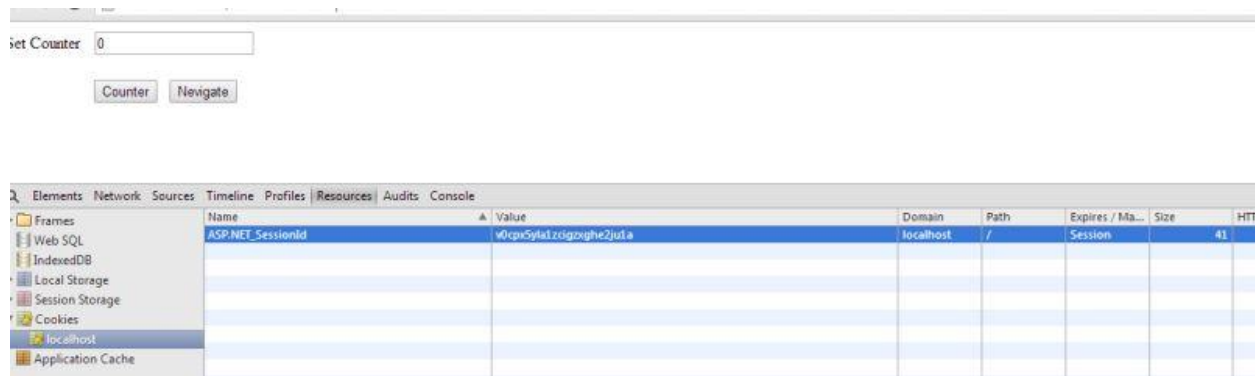
SessionID

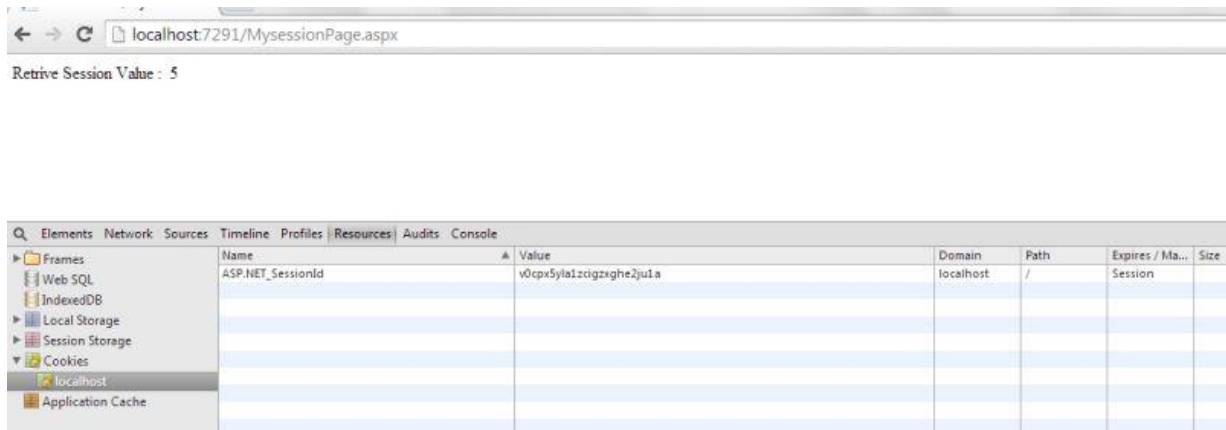
Client Machine Cookies are being used by a session to store a session id. This Session ID is being used by the web server to differentiate among requests of the same user or different ones. SessionID is nothing but a property to uniquely identify a browser with session data on the server. The SessionID value is randomly generated by ASP.NET and stored in a non-expiring session cookie in the browser. The SessionID value is then sent in a cookie with each request to the ASP.NET application.

Definition Reference: MSDN

Dear friends, unfortunately I am not explaining much about cookies here but it is important to spend some time for cookies also.

Cookies are nothing but a small bit of text that accompanies requests and pages as they go between the web server and the browser. The cookie contains information about the web application that can be read whenever the user visits the site.





If the session mode is Inproc and cookieless=true then:

1. `<sessionState mode="InProc" cookieless="true" customProvider="DefaultSessionProvider" >`

The Session id will be sent as part of the URL in every request and if the session id is removed or changed it will be taken as a new request.

`http://localhost:7291/(S(hkjdkowaucoyytjffgha41ab))/Default.aspx`



SessionID

Session Modes

Session modes explain how session variables are being stored, in other words what storage type is used by the variable and what is their behavior.

The default behavior is in memory in an ASP .NET worker process.



Here are the various session modes available in ASP.NET.

- Off
- InProc
- StateServer
- SQLServer
- Custom

Each mode has a different behavior in a web application. They have their own advantages and disadvantages.

Guys, It is very important to understand about the session modes when you are working with an ASP.NET application with session variables as state management techniques. The session modes selected as mode in webconfig enables the ways session variable are stored and it will be then responsible for the application behavior.

Off

If an application has no requirement or need for session state then it's very important to use the off mode. By using this application performance will be better.

InProc Mode

InProc mode can be done in an ASP.NET web application using a configuration file by setting the mode attribute in the element SessionState.

```
1. <sessionState mode="InProc" customProvider="DefaultSessionProvider">
```

When Inproc session state is used the session variables are being stored on the web server memory inside the ASP.NET worker process.

Confused?

Here is the MSDN Definition that says:

ASP.NET runs within a process known as the ASP.NET worker process. All ASP.NET functionality runs within the scope of this process.

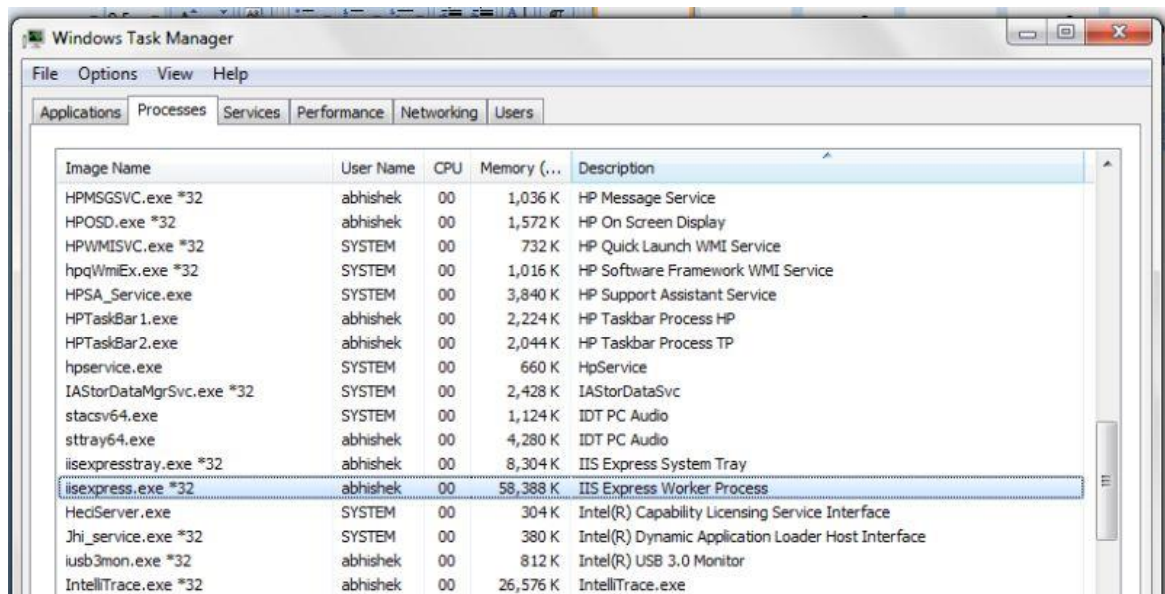


Image Name	User Name	CPU	Memory (...)	Description
HPMSGVSC.exe *32	abhishek	00	1,036 K	HP Message Service
HPOSD.exe *32	abhishek	00	1,572 K	HP On Screen Display
HPWMISVC.exe *32	SYSTEM	00	732 K	HP Quick Launch WMI Service
hpqWmiEx.exe *32	SYSTEM	00	1,016 K	HP Software Framework WMI Service
HPSA_Service.exe	SYSTEM	00	3,840 K	HP Support Assistant Service
HPTaskBar1.exe	abhishek	00	2,224 K	HP Taskbar Process HP
HPTaskBar2.exe	abhishek	00	2,044 K	HP Taskbar Process TP
hpservice.exe	SYSTEM	00	660 K	HpService
IAStorDataMgrSvc.exe *32	SYSTEM	00	2,428 K	IAStorDataSvc
stacsv64.exe	SYSTEM	00	1,124 K	IDT PC Audio
stray64.exe	abhishek	00	4,280 K	IDT PC Audio
iisexpress.exe *32	abhishek	00	8,304 K	IIS Express System Tray
iisexpress.exe *32	abhishek	00	58,388 K	IIS Express Worker Process
HedServer.exe	SYSTEM	00	304 K	Intel(R) Capability Licensing Service Interface
Jhi_service.exe *32	SYSTEM	00	380 K	Intel(R) Dynamic Application Loader Host Interface
iusb3mon.exe *32	abhishek	00	812 K	Intel(R) USB 3.0 Monitor
IntelliTrace.exe *32	abhishek	00	26,576 K	IntelliTrace.exe

Basically session variables are stored in the executable that is an IIS worker process. So if the request is coming from the same user the data will be available.

Advantages of InProc:

- Easy to Implement.
- Complex Objects can be added without serialization.
- Best in performance compared to out-of-process modes.

Disadvantages of InProc :

- Not able to sustain the session values when the worker process/IIS is restarted. In that case data loss will happen which make the application break.
- Scalability is a major problem.
- Not good for applications with a large user base.

Where to Use

In Proc mode is best suited for the application that is hosted on a single server and mid size use base or the session variable used is not big, to avoid data loss and scalability issues. When there is a requirement for a web farm or web garden deployments the “out of process “modes like state server or SQL Server modes are the best option.

State Server Session Mode

The disadvantage of session data loss is due to the worker process recycle that can be reduced using another mode, the state server mode.

Reference MSDN Definition

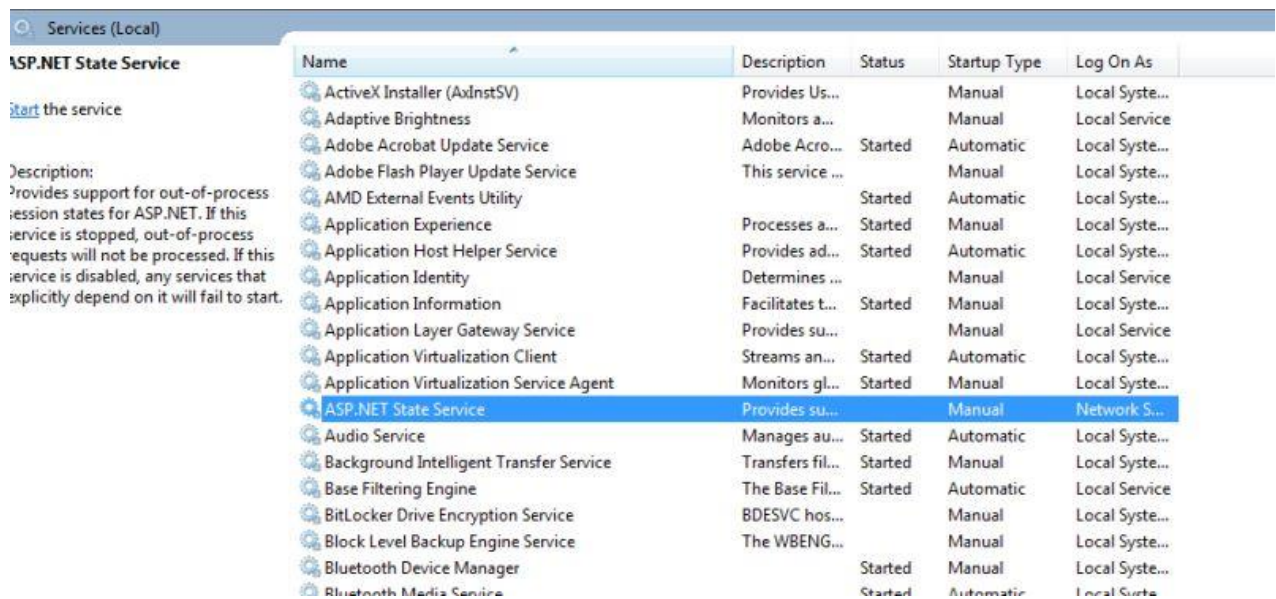
StateServer mode, that stores session state in a separate process called the ASP.NET state service. This ensures that session state is preserved if the web application is restarted and also makes session state available to multiple Web servers in a Web farm.

ASP.NET is a Windows services that stores the session variable data in their process.

Procedure to set up state server mode

Go to Run then enter "Services.msc" then Start ASP.NET State Service.

By default ASP.NET state service is in manual mode.



1. `<sessionState mode="StateServer" customProvider="DefaultSessionProvider" stateConnectionString="tcpip=localhost:42424">`
- 2.
3. `stateConnectionString="tcpip=localhost : 42424"`



Server Name/IP Port

StateConnectionString basically consists of the following;

Application State

The MSDN Definition says: Application state is a data repository available to all classes in an ASP.NET application. Application state is stored in memory on the server and is faster than storing and retrieving information in a database. Unlike session state, which is specific to a single user session, application state applies to all users and sessions.

Application state is stored in an instance of the `HttpApplicationState` class. This class exposes a key-value dictionary of objects.

Application state variables are also used to store data when navigating from one page to another. It's multi-user Global data meaning it will be accessible across all pages and all sessions. Application state variables are stored on the web server in ASP.NET worker process memory.

Sample Code

Addition of data in application variables.

```

1. namespace Test
2. {
3.     public partial class applicationState : System.Web.UI.Page
4.     {
5.         protected void Page_Load(object sender, EventArgs e)
6.         {
7.             if (!IsPostBack)
8.             {
9.                 if (Application["Counter"] == null)
10.                {
11.                    Application["Counter"] = 0;
12.                }
13.                TextBox1.Text = Application["Counter"].ToString();
14.            }
15.        }
16.    }
17.
18.    protected void Button1_Click(object sender, EventArgs e)
19.    {
20.        if (Application["Counter"] != null)
21.        {
22.            int ApplicationCounter = (int)Application["Counter"] + 1;
23.            TextBox1.Text = ApplicationCounter.ToString();
24.            Application["Counter"] = ApplicationCounter;
25.        }
26.    }
27.
28.
29.    protected void Button2_Click(object sender, EventArgs e)
30.    {
31.        Response.Redirect("ApplicationStateTest.aspx");
32.    }
33. }
34. }

```

Reading data from application variables:

```

1. namespace Test
2. {
3.     public partial class ApplicationStateTest : System.Web.UI.Page
4.     {
5.         protected void Page_Load(object sender, EventArgs e)
6.         {

```

```

7.         if (Application["Counter"] != null)
8.         {
9.             Label1.Text = Application["Counter"].ToString();
10.        }
11.    }
12. }
13. }

```

An application state variable does not have any default time unlike session variables. It will end when the application hosting process is restarted or the application ends.

Application state variable data is not reliable for web farms or web garden type deployment since they are not shared across multiple web servers or multiple worker processes on the same machine so using application variable data could cause the problem of data loss and the application breaks.

An application state variable is not thread safe, meaning multiple users from a different session can access the variable and can manipulate the variable.

To ensure the data integrity and resolve concurrency issues while using application state variables, lock and unlock methods should be used.

Important Note

Web farm: It's a situation where the web application is deployed on different servers with load balancer.

Web garden: It's a situation where the web application is deployed on the same server with multiple worker processes.

Advantages of application state:

- Application variable data is multi-user global data stored in memory.
- Easy to access.
- Fast retrieval.

Disadvantages of application state:

- Application variable data is not able to survive the IIS restart and worker process recycling.
- Not suited for web farm and web garden like deployment situation.

Where to Use

An application variable is used only when the variable needs to have global access and when you need them for the entire time, during the lifetime of an application.

Cookies is a small piece of information stored on the client machine. This file is located on client machines "C:\Document and Settings\Currently_Login user\Cookie" path. Its is used to store user preference information like Username, Password, City

and PhoneNo etc on client machines. We need to import namespace called `System.Web.HttpCookie` before we use cookie.

Type of Cookies

1. Persist Cookie - A cookie has not have expired time Which is called as Persist Cookie
2. Non-Persist Cookie - A cookie has expired time Which is called as Non-Persist Cookie

How to create a cookie?

It is really easy to create a cookie in the Asp.Net with help of Response object or `HttpCookie`

Example 1

```
1. HttpCookie userInfo = new HttpCookie("userInfo");
2. userInfo["UserName"] = "Annathurai";
3. userInfo["UserColor"] = "Black";
4. userInfo.Expires.Add(new TimeSpan(0, 1, 0));
5. Response.Cookies.Add(userInfo);
```

Example 2

```
1. Response.Cookies["userName"].Value = "Annathurai";
2. Response.Cookies["userColor"].Value = "Black";
```

How to retrieve from cookie?

It is easy way to retrieve cookie value form cookies with the help of Request object.

Example 1

```
1. string User_Name = string.Empty;
2. string User_Color = string.Empty;
3. User_Name = Request.Cookies["userName"].Value;
4. User_Color = Request.Cookies["userColor"].Value;
```

Example 2

```
1. string User_name = string.Empty;
2. string User_color = string.Empty;
3. HttpCookie reqCookies = Request.Cookies["userInfo"];
4. if (reqCookies != null)
5. {
6.     User_name = reqCookies["UserName"].ToString();
7.     User_color = reqCookies["UserColor"].ToString();
8. }
```


Cookie's

When we make a request from the client to web server, the web server process the request and give a lot of information with big packets which will have Header information, Metadata, cookies etc., Then repose object can do all the things with browser.

Cookie's common property

1. Domain => Which is used to associate cookies to domain.
2. Secure => We can enable secure cookie to set true(HTTPs).
3. Value => We can manipulate individual cookie.
4. Values => We can manipulate cookies with key/value pair.
5. Expires => Which is used to set expire date for the cookies.

Advantages of Cookie

1. Its clear text so user can able to read it.
2. We can store user preference information on the client machine.
3. Its easy way to maintain.
4. Fast accessing.

Disadvantages of Cookie

1. If user clear cookie information we can't get it back.
2. No security.
3. Each request will have cookie information with page.

How to clear the cookie information?

1. We can clear cookie information from client machine on cookie folder
2. To set expires to cookie object
 1. `userInfo.Expires = DateTime.Now.AddHours(1);`

It will clear the cookie with one hour duration.

What is global.asax

IsPostBack

Global.asax is an optional file which is used to handling higher level application events such as `Application_Start`, `Application_End`, `Session_Start`, `Session_End` etc. It is also popularly known as ASP.NET Application File. This file resides in the root directory of an ASP.NET-based application.

Global.asax contains a Class representing your application as a whole. At run time, this file is parsed and compiled into a dynamically

generated .NET Framework class derived from the `HttpApplication` base class. You can deploy this file as an assembly in the `\bin` directory of an ASP.NET application. The `Global.asax` file itself is configured so that if a user requests the file, the request is rejected. External users cannot download or view the code written within it.

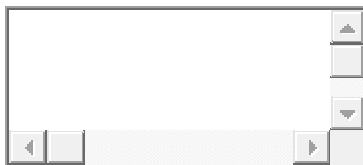
How to create a Global.asax file

`Global.asax` file doesn't create normally; you need to add it by yourself.

How to ?

Open Visual Studio
Create a new website
Go to the Solution Explorer
Add New Item
Global Application Class
Add

Your `Global.asax` file looks like this



After that you need to add a class in your project.

Right clicking `App_Code`
Add New Item
Class
name it `Global.cs`
Add

Inherit the newly generated by `System.Web.HttpApplication` and copy all the methods created `Global.asax` to `Global.cs` and also add an `inherit` attribute to the `Global.asax` file

Now your `Global.asax` will look like following:

```

public class Global : System.Web.HttpApplication
{
    public Global()
    {
        //
        // TODO: Add constructor logic here
        //
    }

    void Application_Start(object sender, EventArgs e)
    {
        // Code that runs on application startup

    }
    /// Many other events like begin request...e.t.c, e.t.c
}

```

What is Caching?

Caching is a technique of storing frequently used data/information in memory, so that, when the same data/information is needed next time, it could be directly retrieved from the memory instead of being generated by the application.

Caching is extremely important for performance boosting in ASP.NET, as the pages and controls are dynamically generated here. It is especially important for data related transactions, as these are expensive in terms of response time.

Caching places frequently used data in quickly accessed media such as the random access memory of the computer. The ASP.NET runtime includes a key-value map of CLR objects called cache. This resides with the application and is available via the HttpContext and System.Web.UI.Page.

In some respect, caching is similar to storing the state objects. However, the storing information in state objects is deterministic, i.e., you can count on the data being stored there, and caching of data is nondeterministic.

The data will not be available in the following cases:

- If its lifetime expires,
- If the application releases its memory,
- If caching does not take place for some reason.

You can access items in the cache using an indexer and may control the lifetime of objects in the cache and set up links between the cached objects and their physical sources.

Caching in ASP.Net

ASP.NET provides the following different types of caching:

- **Output Caching** : Output cache stores a copy of the finally rendered HTML pages or part of pages sent to the client. When the next client requests for this page, instead of regenerating the page, a cached copy of the page is sent, thus saving time.
- **Data Caching** : Data caching means caching data from a data source. As long as the cache is not expired, a request for the data will be fulfilled from the cache. When the cache is expired, fresh data is obtained by the data source and the cache is refilled.
- **Object Caching** : Object caching is caching the objects on a page, such as data-bound controls. The cached data is stored in server memory.
- **Class Caching** : Web pages or web services are compiled into a page class in the assembly, when run for the first time. Then the assembly is cached in the server. Next time when a request is made for the page or service, the cached assembly is referred to. When the source code is changed, the CLR recompiles the assembly.
- **Configuration Caching** : Application wide configuration information is stored in a configuration file. Configuration caching stores the configuration information in the server memory.

In this tutorial, we will consider output caching, data caching, and object caching.

Output Caching

Rendering a page may involve some complex processes such as, database access, rendering complex controls etc. Output caching allows bypassing the round trips to server by caching data in memory. Even the whole page could be cached.

The `OutputCache` directive is responsible of output caching. It enables output caching and provides certain control over its behaviour.

Syntax for `OutputCache` directive:

```
<% @ OutputCache Duration="15" VaryByParam="None" %>
```

Put this directive under the page directive. This tells the environment to cache the page for 15 seconds. The following event handler for page load would help in testing that the page was really cached.

```
protected void Page_Load(object sender, EventArgs e)
{
    Thread.Sleep(10000);
    Response.Write("This page was generated and cache at:" +
        DateTime.Now.ToString());
}
```

The **`Thread.Sleep()`** method stops the process thread for the specified time. In this example, the thread is stopped for 10 seconds, so when the page is loaded for first time, it takes 10 seconds. However, next time you refresh the page it does not take any time, as the page is retrieved from the cache without being loaded.

The `OutputCache` directive has the following attributes, which helps in controlling the behaviour of the output cache:

Attribute	Values	Description
DiskCacheable	true/false	Specifies that output could be written to a disk based cache.
NoStore	true/false	Specifies that the "no store" cache control header is sent or not.
CacheProfile	String name	Name of a cache profile as to be stored in web.config.
VaryByParam	None * Param- name	Semicolon delimited list of string specifies query string values in a GET request or variable in a POST request.
VaryByHeader	* Header names	Semicolon delimited list of strings specifies headers that might be submitted by a client.
VaryByCustom	Browser Custom string	Tells ASP.NET to vary the output cache by browser name and version or by a custom string.
Location	Any Client Downstream Server None	Any: page may be cached anywhere. Client: cached content remains at browser. Downstream: cached content stored in downstream and server both. Server: cached content saved only on server. None: disables caching.
Duration	Number	Number of seconds the page or control is cached.

Let us add a text box and a button to the previous example and add this event handler for the button.

```
protected void btnmagic_Click(object sender, EventArgs e)
{
    Response.Write("<br><br>");
    Response.Write("<h2> Hello, " + this.txtname.Text + "</h2>");
}
```

Change the OutputCache directive:

```
<% @ OutputCache Duration="60" VaryByParam="txtname" %>
```

When the program is executed, ASP.NET caches the page on the basis of the name in the text box.

Data Caching

The main aspect of data caching is caching the data source controls. We have already discussed that the data source controls represent data in a data source, like a database or an XML file. These controls derive from the abstract class DataSourceControl and have the following inherited properties for implementing caching:

- **CacheDuration** - It sets the number of seconds for which the data source will cache data.
- **CacheExpirationPolicy** - It defines the cache behavior when the data in cache has expired.
- **CacheKeyDependency** - It identifies a key for the controls that auto-expires the content of its cache when removed.
- **EnableCaching** - It specifies whether or not to cache the data.

Example

To demonstrate data caching, create a new website and add a new web form on it. Add a SqlDataSource control with the database connection already used in the data access tutorials.

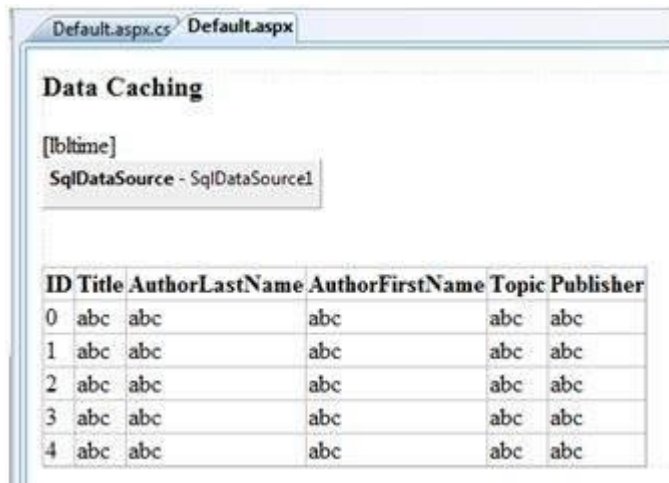
For this example, add a label to the page, which would show the response time for the page.

```
<asp:Label ID="lbltime" runat="server"></asp:Label>
```

Apart from the label, the content page is same as in the data access tutorial. Add an event handler for the page load event:

```
protected void Page_Load(object sender, EventArgs e)
{
    lbltime.Text = String.Format("Page posted at: {0}", DateTime.Now.ToLongTimeString());
}
```

The designed page should look as shown:



When you execute the page for the first time, nothing different happens, the label shows that, each time you refresh the page, the page is reloaded and the time shown on the label changes.

Next, set the EnableCaching attribute of the data source control to be 'true' and set the Cacheduration attribute to '60'. It will implement caching and the cache will expire every 60 seconds.

The timestamp changes with every refresh, but if you change the data in the table within these 60 seconds, it is not shown before the cache expires.

```
<asp:SqlDataSource ID = "SqlDataSource1" runat = "server"
  ConnectionString = "<%"$ ConnectionStrings: ASPDotNetStepByStepConnectionString %>"
  ProviderName = "<%"$ ConnectionStrings:
  ASPDotNetStepByStepConnectionString.ProviderName %>"
  SelectCommand = "SELECT * FROM [DotNetReferences]"
  EnableCaching = "true" CacheDuration = "60">
</asp:SqlDataSource>
```

Object Caching

Object caching provides more flexibility than other cache techniques. You can use object caching to place any object in the cache. The object can be of any type - a data type, a web control, a class, a dataset object, etc. The item is added to the cache simply by assigning a new key name, shown as follows Like:

```
Cache["key"] = item;
```

ASP.NET also provides the Insert() method for inserting an object to the cache. This method has four overloaded versions. Let us see them:

Overload	Description
Cache.Insert((key, value);	Inserts an item into the cache with the key name and value with default

	priority and expiration.
Cache.Insert(key, value, dependencies);	Inserts an item into the cache with key, value, default priority, expiration and a CacheDependency name that links to other files or items so that when these change the cache item remains no longer valid.
Cache.Insert(key, value, dependencies, absoluteExpiration, slidingExpiration);	This indicates an expiration policy along with the above issues.
Cache.Insert(key, value, dependencies, absoluteExpiration, slidingExpiration, priority, onRemoveCallback);	This along with the parameters also allows you to set a priority for the cache item and a delegate that, points to a method to be invoked when the item is removed.

Sliding expiration is used to remove an item from the cache when it is not used for the specified time span. The following code snippet stores an item with a sliding expiration of 10 minutes with no dependencies.

```
Cache.Insert("my_item", obj, null, DateTime.MaxValue, TimeSpan.FromMinutes(10));
```

Example

Create a page with just a button and a label. Write the following code in the page load event:

```
protected void Page_Load(object sender, EventArgs e)
{
    if (this.IsPostBack)
```



```

{
    lblinfo.Text += "Page Posted Back.<br/>";
}
else
{
    lblinfo.Text += "page Created.<br/>";
}

if (Cache["testitem"] == null)
{
    lblinfo.Text += "Creating test item.<br/>";
    DateTime testItem = DateTime.Now;
    lblinfo.Text += "Storing test item in cache ";
    lblinfo.Text += "for 30 seconds.<br/>";
    Cache.Insert("testitem", testItem, null,
        DateTime.Now.AddSeconds(30), TimeSpan.Zero);
}
else
{
    lblinfo.Text += "Retrieving test item.<br/>";
    DateTime testItem = (DateTime)Cache["testitem"];
    lblinfo.Text += "Test item is: " + testItem.ToString();
    lblinfo.Text += "<br/>";
}

lblinfo.Text += "<br/>";
}

```

When the page is loaded for the first time, it says:

Page Created.

Creating test item.

Storing test item in cache for 30 seconds.

If you click on the button again within 30 seconds, the page is posted back but the label control gets its information from the cache as shown:

Page Posted Back.

Retrieving test item.

Test item is: 14-07-2010 01:25:04

ASP.NET UNIT-3

ASP.NET DataGrid

.NET Framework provides DataGrid control to display data on the web page. It was introduced in .NET 1.0 and now has been deprecated. DataGrid is used to display data in scrollable grid. It requires data source to populate data in the grid.

It is a server side control and can be dragged from the toolbox to the web form. Data Source for the DataGrid can be either a DataTable or a database. Let's see an example, how can we create a DataGrid in our application.

This tutorial contains two examples. One is using the DataTable and second is using the database to display data into the DataGrid.

ASP.NET DataGrid Example with DataTable

This example uses DataTable to bind data to the DataGrid control.

Play Video

// DataGridExample2.aspx

1. `<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="DataGridExample2.aspx.cs" Inherits="DataGridExample.DataGridExample2" %>`
2. `<!DOCTYPE html>`
3. `<html xmlns="http://www.w3.org/1999/xhtml">`
4. `<head runat="server">`
5. `<title></title>`
6. `</head>`
7. `<body>`
8. `<form id="form1" runat="server">`
9. `<div>`

```
10.     <p>This DataGrid contains DataTable records </p>
11.     <asp:DataGrid ID="DataGrid1" runat="server">
12.     </asp:DataGrid>
13.     </div>
14. </form>
15. </body>
16. </html>
```

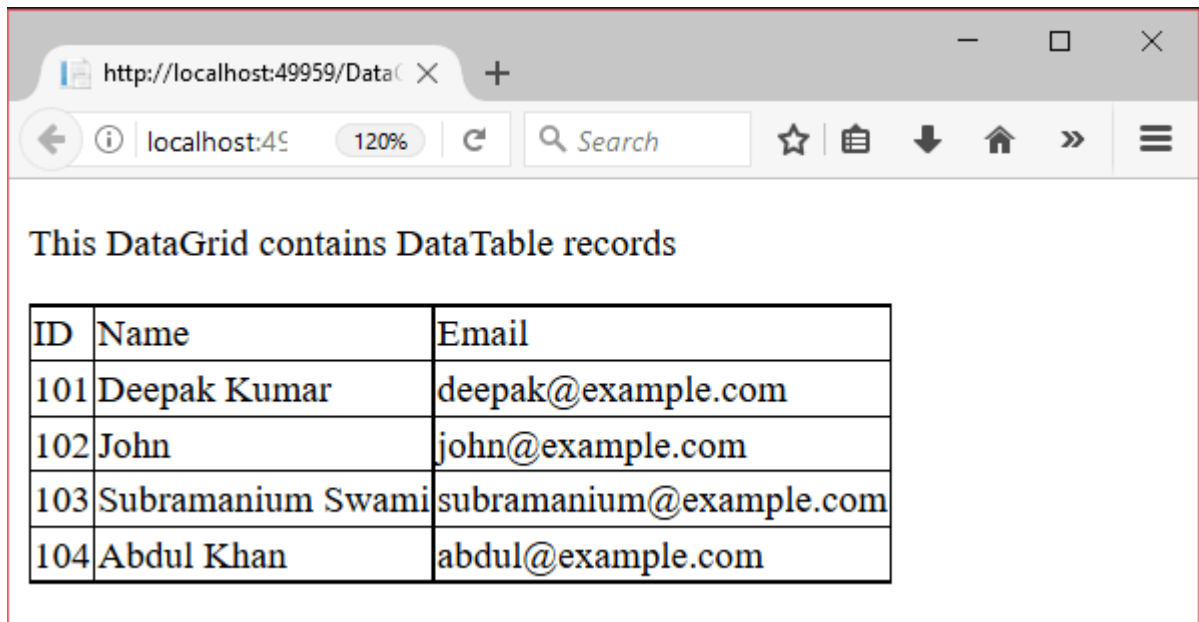
CodeBehind

// DataGridExample2.aspx.cs

```
1. using System;
2. using System.Data;
3. namespace DataGridExample
4. {
5.     public partial class DataGridExample2 : System.Web.UI.Page
6.     {
7.         protected void Page_Load(object sender, EventArgs e)
8.         {
9.             DataTable table = new DataTable();
10.            table.Columns.Add("ID");
11.            table.Columns.Add("Name");
12.            table.Columns.Add("Email");
13.            table.Rows.Add("101", "Deepak Kumar", "deepak@example.com");
14.            table.Rows.Add("102", "John", "john@example.com");
15.            table.Rows.Add("103", "Subramanium Swami", "subramanium@example.com"
16.            );
17.            table.Rows.Add("104", "Abdul Khan", "abdul@example.com");
18.            DataGrid1.DataSource = table;
19.            DataGrid1.DataBind();
20.        }
21.    }
```

Output:

It produces the following output to the browser.

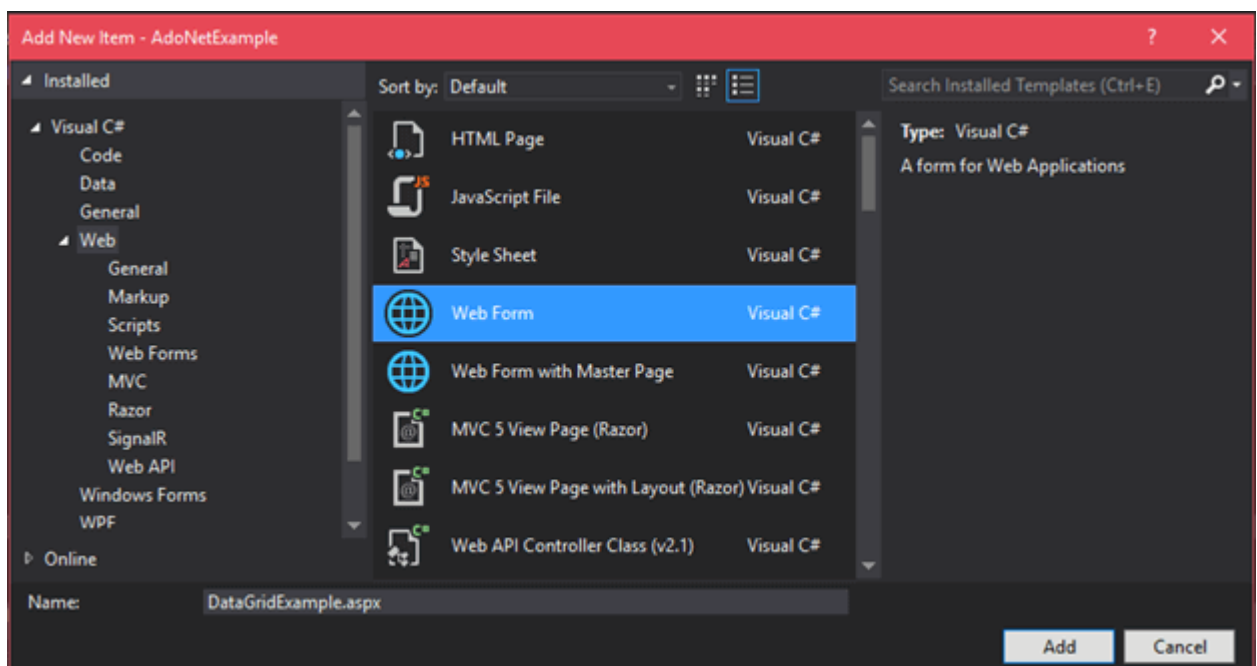


ASP.NET DataGrid Example with Database

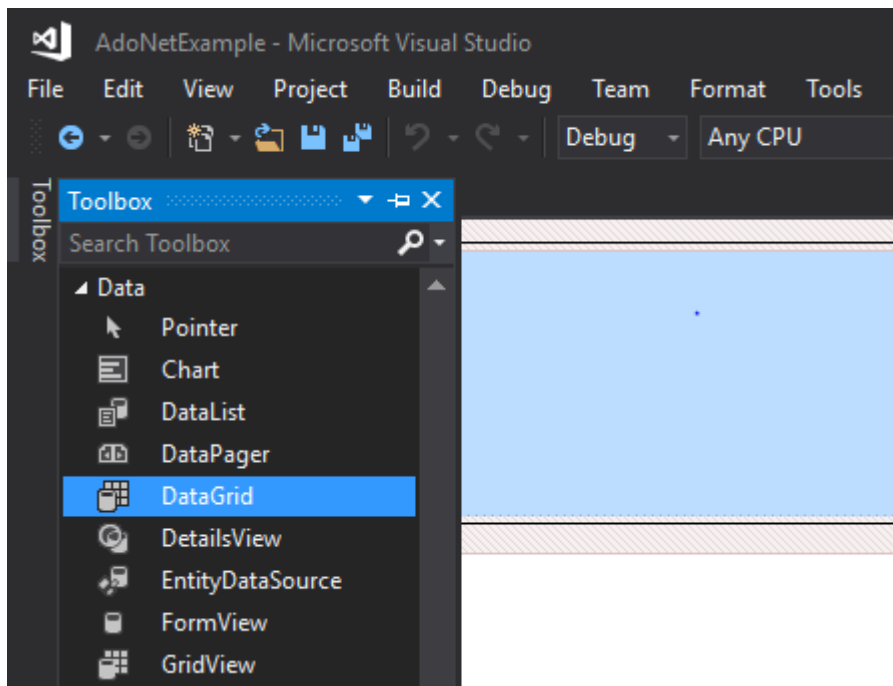
This example uses database as a data source to display data to the DataGrid. This example includes the following steps.

1) Add a web form

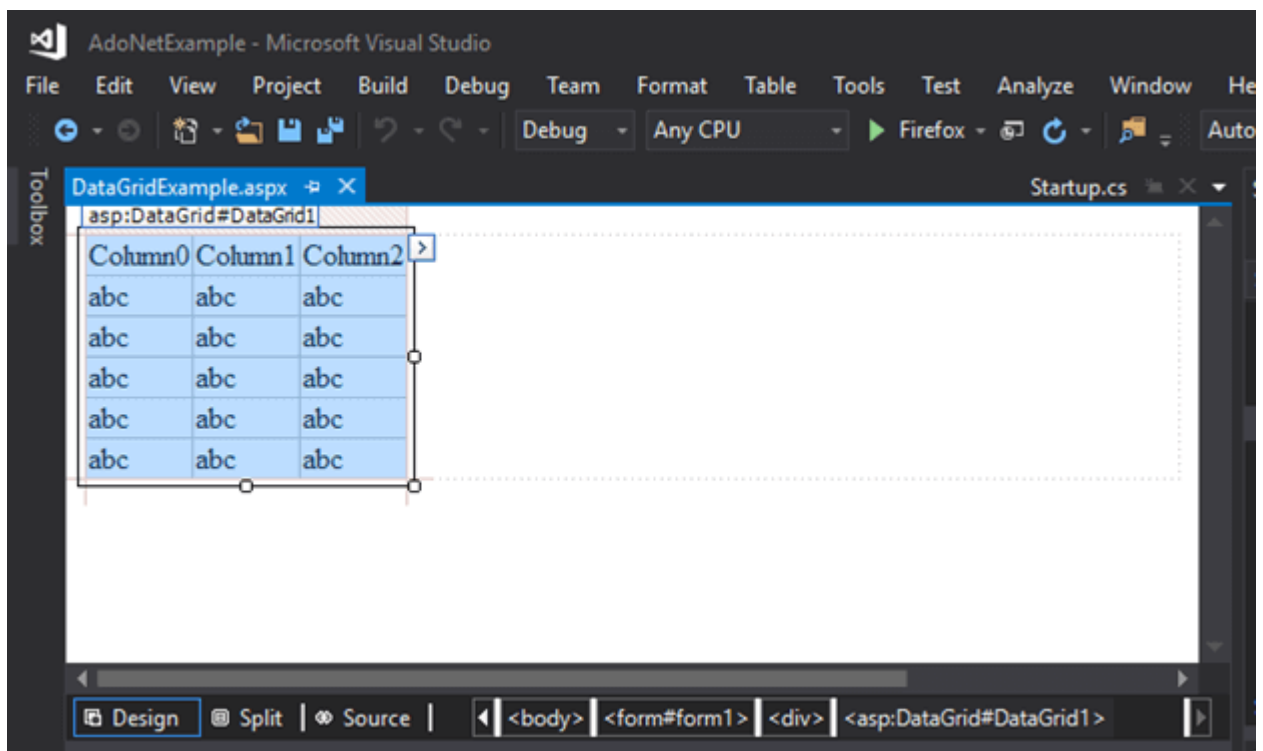
Create a new form to drag DataGrid upon it. See, as we did in the following screenshot.



After adding, now, open toolbox and drag DataGrid control to the form.



After dragging, initially it looks like the following.



This form contains the following source code at backend.

// DataGridExample.aspx

1. <%@ Page Language="C#" AutoEventWireup="true"
2. CodeBehind="DataGridExample.aspx.cs" Inherits="AdoNetExample.DataGridExample"
3. %>
4. <!DOCTYPE html>
5. <html xmlns="http://www.w3.org/1999/xhtml">
6. <head runat="server">
7. <title></title>
8. </head>
9. <body>
10. <form id="form1" runat="server">
11. <div>
12. </div>
13. <asp:DataGrid ID="DataGrid1" runat="server">
14. </asp:DataGrid>
15. </form>
16. </body>
17. </html>

2) Connect to the Database

In the CodeBehind file, we have code of database connectivity, and binding fetched record to the DataGrid.

CodeBehind file

// DataGridExample.aspx.cs

1. using System;
2. using System.Data;
3. using System.Data.SqlClient;
4. namespace AdoNetExample
5. {
6. public partial class DataGridExample : System.Web.UI.Page
7. {
8. protected void Page_Load(object sender, EventArgs e)
9. {
10. using (SqlConnection con = new SqlConnection("data source=.; database=student; integrated security=SSPI"))

```

11.      {
12.          SqlDataAdapter sde = new SqlDataAdapter("Select * from student", con);
13.          DataSet ds = new DataSet();
14.          sde.Fill(ds);
15.          DataGrid1.DataSource = ds;
16.          DataGrid1.DataBind();
17.      }
18.  }
19.  }
20.}

```

Records in SQL Server Table

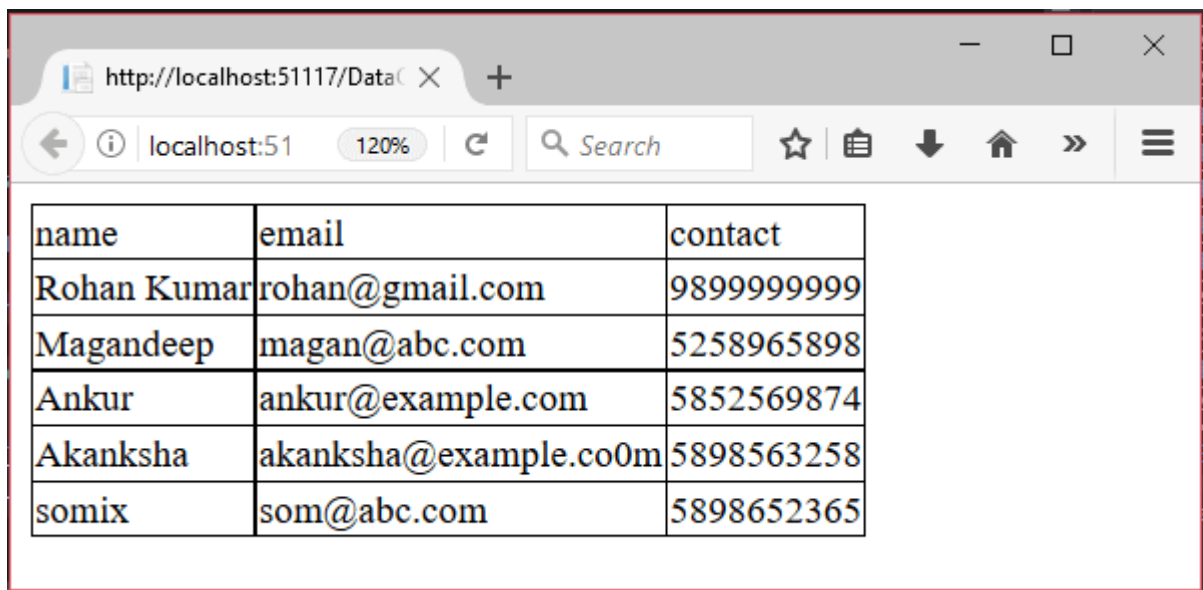
A **student** table contains records that we want to display by using the DataGrid. This table contains the following records.

The screenshot shows the Microsoft SQL Server Management Studio interface. On the left, the Object Explorer displays the database structure for 'DESKTOP-EDFPJEL (SQL Server 10.50.4000 - DES)'. The 'Student' database is expanded, showing 'Tables' and 'Columns'. The 'dbo.student' table is selected, and its columns are listed: 'name (nvarchar(100), null)', 'email (nvarchar(50), null)', and 'contact (nvarchar(12), null)'. On the right, the SQL Query window shows the query 'select * from student'. Below the query, the Results pane displays the data from the 'student' table in a table format.

	name	email	contact
1	Rohan Kumar	rohan@gmail.com	9899999999
2	Magandeep	magan@abc.com	5258965898
3	Ankur	ankur@example.com	5852569874
4	Akanksha	akanksha@example.co0m	5898563258
5	somix	som@abc.com	5898652365

Output:

After executing this application, it fetches records from the SQL server and displays to the web browser.



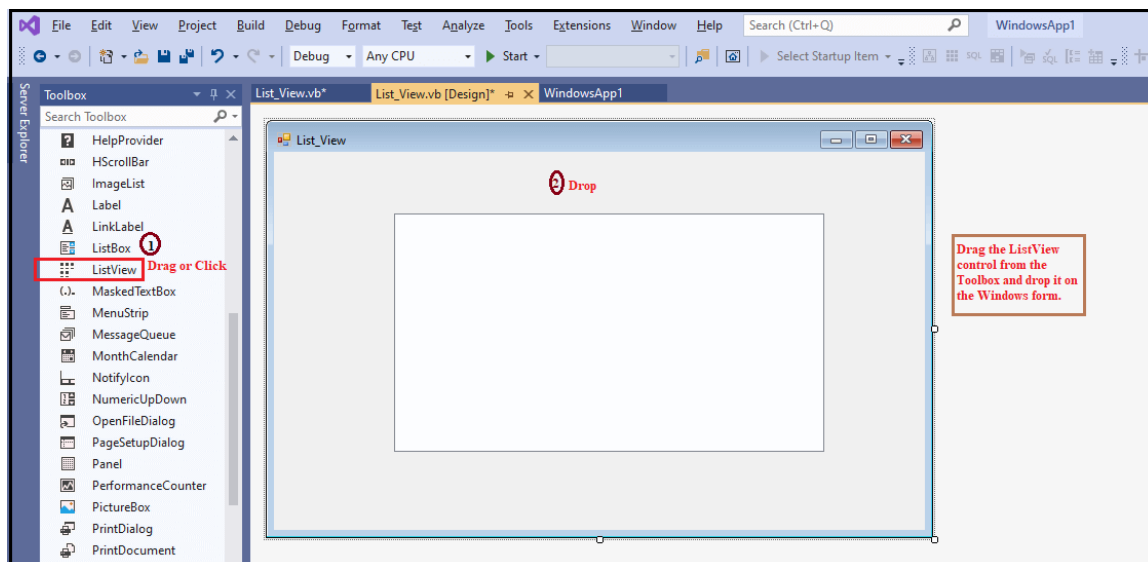
name	email	contact
Rohan Kumar	rohan@gmail.com	9899999999
Magandeep	magan@abc.com	5258965898
Ankur	ankur@example.com	5852569874
Akanksha	akanksha@example.co0m	5898563258
somix	som@abc.com	5898652365

VB.NET ListView Control

The ListView Controls are used to display a collection of items in the Windows Forms. It uses one of the view lists, such as LargeIcon, SmallIcon, Details, List, and Tile. Furthermore, the ListView allows the user to add or remove icons from the ListView Control.

Let's create a ListView control in the [VB.NET Windows](#) form by using the following steps.

Step 1: We have to find the ListView control from the toolbox and then drag and drop the ListView control onto the window form, as shown below.



Step 2: Once the ListView is added to the form, we can set various properties of the List by clicking on the ListView control.

Properties of the ListView Control

Properties	Description
Alignment	The Alignment property is used to set the alignment for the item in the ListView Control.
Activation	The Activation property is used to get or set a user-requested action to activate an item.
CheckBoxes	The CheckBoxes property sets a value to display the checkbox that appears next to each item in the list view control.
Columns	The Columns property is used to display the collection of columns header in the ListView Control.
CheckIndices	The CheckIndices property is used to get all checked items in the ListView Control.
GridLines	The GridLines Property is used to display the gridlines between the rows and columns that contain the items and subitems in the ListView Control.
Items	It is used to collect the collection of the items that are in the ListView Control.
LargeImageList	It is used to set or get ImageList to display it as a large icon in the ListView Control.
HeaderStyle	It is used to set or get the column header style in the ListView control.

MultiSelect	The MultiSelect property is used to set a value that allows selecting more than items in the ListView control.
SelectedItems	It is used to obtain all selected items in the ListView control.
ShowGroups	The ShowGroups property set a value that represents whether the ListView items are displayed in groups.
SmallImageList	It is used to set or get ImageList to display the image as a small icon in the ListView control.
TopItem	The TopItem property is used to set or get the first item in control.
View	<p>The View property is used to set the items displayed in the List view. The View property has the following values:</p> <p>SmallIcon: It is used to display small size icons.</p> <p>List: It is used to display items in a list, and it only shows captions.</p> <p>LargeIcon: It is used to display large size icons.</p> <p>Report: It is used to display items and its sub-items.</p>

There are following properties of ListView control.

Method	Description
Arrangelcons()	The Arrangelcons method is used to arrange all the items displayed as icons in the ListView control.
FindItemWithText()	It is used to search the first ListViewItem that started with the given text value.
GetItemAt()	The GetItemAt method is used to get an item at the specified location of the ListView control.
Clear()	The Clear method is used to clear all the items and columns from the ListView control.
Sort()	The Sort method is used to sort items of the ListView.

Methods of the ListView Control

Events of the ListView Control

There are following events of the ListView control.

Events	Description
ItemActivate	The ItemActivate event occurred when an item activated in the ListView control.
ItemChecked	The ItemChecked event is found in the ListView control when the checked state of an item changes.
TextChanged	The TextChanged event is found in the ListView control when the property of the text changes.
ItemCheck	When the status of a check item changes, the ItemCheck event is found in the list view control.
AfterLabelEditEvent	It occurs when the user in the ListView control edits the label for an item.

Let's create a program to insert the records in the ListView control of the VB.NET Windows form.

List_View.vb

1. Public Class List_View
2. Private Sub List_View_Load(sender As Object, e As EventArgs) Handles MyBase.Load
3. Me.Text = "javatpoint.com" 'Set the title for the Windows form
4. ListView1.View = View.Details ' Display the List in details
5. ListView1.GridLines = True ' Set the Grid lines
6. Label1.Text = "Enter the Roll No." ' Set the name of Labels
7. Label2.Text = "Enter Your Name"
8. Label3.Text = "Enter Your Email"
9. Label4.Text = "Enter the Course"
10. Label5.Text = "Student details"
11. ListView1.Columns.Add("Roll No", 70, HorizontalAlignment.Left) ' set the name of column
12. ListView1.Columns.Add("Name", 100, HorizontalAlignment.Left) ' set the name of column
13. ListView1.Columns.Add("Email", 150, HorizontalAlignment.Left) ' set the name of column
14. ListView1.Columns.Add("Course", 100, HorizontalAlignment.Left) ' set the name of column
15. ListView1.BackColor = Color.LightSkyBlue

```

16. Button1.Text = "Add New Entry"
17. Button1.ForeColor = Color.White
18. Button1.BackColor = Color.Green
19. Label5.ForeColor = Color.Red
20. Button2.Text = "Exit"
21. Button2.BackColor = Color.Red
22. End Sub
23. Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click

24. Dim str(4) As String
25. Dim itm As ListViewItem
26. str(0) = TextBox1.Text 'Accept value from the user.
27. str(1) = TextBox2.Text
28. str(2) = TextBox3.Text
29. str(3) = TextBox4.Text
30. itm = New ListViewItem(str)
31. ListView1.Items.Add(itm) 'Add the items into the ListView
32. End Sub
33. Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click

34. End 'Exit from the program
35. End Sub
36. End Class

```

Output:

The screenshot shows a Windows application window with the title bar 'javatpoint.com'. The main content area has a light gray background. At the top center, the text 'Student details' is displayed in red. On the left side, there are four text boxes stacked vertically, each preceded by a label: 'Enter the Roll No.', 'Enter Your Name', 'Enter Your Email', and 'Enter the Course'. To the right of these text boxes is a table with four columns: 'Roll No', 'Name', 'Email', and 'Course'. The table has 10 empty rows. At the bottom left, there is a green button with the text 'Add New Entry'. At the bottom right, there is a red button with the text 'Exit'.

Now, we fill all the details of the Student that is asked in the Form.

[illegible]

Now, click on the **Add New Entry** button. It displays the record in the ListView control or in Student details table, as shown below.

Student details

Enter the Roll No.
101

Enter Your Name
James

Enter Your Email
abc@example.com

Enter the Course
Comp. Program

Roll No	Name	Email	Course
101	James	abc@example.com	Comp. Program

Add New Entry **Exit**

Similarly, we have added the following student details in the form.

The screenshot shows a web application window titled 'javatpoint.com'. The main heading is 'Student details' in red. On the left, there are four input fields with labels: 'Enter the Roll No.' (containing '107'), 'Enter Your Name' (containing 'Rose'), 'Enter Your Email' (containing 'abc@javatpoint'), and 'Enter the Course' (containing 'uman Resource'). Below these fields are two buttons: 'Add New Entry' (green) and 'Exit' (red). To the right of the form is a table with four columns: 'Roll No', 'Name', 'Email', and 'Course'. The table contains five rows of data, with the last row highlighted in blue.

Roll No	Name	Email	Course
101	James	abc@example.com	Comp. Progr
102	Peter	ab12@example.com	Multimedia
105	Robert	abc121@exam.com	Electronics
106	Maeve	xyz@javatpoint.com	Project mana
107	Rose	abc@javatpoint.com	Human Reso

DetailsView Class

Definition

Namespace:

[System.Web.UI.WebControls](#)

Assembly:

System.Web.dll

Displays the values of a single record from a data source in a table, where each data row represents a field of the record. The [DetailsView](#) control allows you to edit, delete, and insert records.

C#Copy

```
[System.Web.UI.ControlValueProperty("SelectedValue")]  
public class DetailsView : System.Web.UI.WebControls.CompositeDataBoundControl,  
System.Web.UI.ICallbackEventHandler, System.Web.UI.IDataItemContainer,  
System.Web.UI.IPostBackEventHandler, System.Web.UI.WebControls.ICallbackContainer,  
System.Web.UI.WebControls.IDataBoundItemControl,  
System.Web.UI.WebControls.IFieldControl,  
System.Web.UI.WebControls.IPostBackContainer
```

Inheritance

[Object](#)

[Control](#)

[WebControl](#)

[BaseDataBoundControl](#)
[DataBoundControl](#)
[CompositeDataBoundControl](#)
DetailsView

Attributes

[ControlValuePropertyAttribute](#)

Implements

[ICallbackEventHandler](#) [IDataItemContainer](#) [INamingContainer](#) [IPostBackEventHandler](#) [ICallbackContainer](#) [IPostBackContainer](#) [IDataBoundControl](#) [IDataBoundItemControl](#) [IFieldControl](#)

Examples

The following code example demonstrates how to use a [DetailsView](#) control in combination with a [GridView](#) control for a simple master-detail scenario. It displays the details of an item selected in the [GridView](#) control.

ASP.NET (C#)Copy

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>ASP.NET Example</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <table>
                <tr>
                    <td>
                        <asp:GridView ID="GridView1" runat="server"
                            AutoGenerateColumns="False" DataSourceID="Customers"
                            DataKeyNames="CustomerID">
                            <Columns>
                                <asp:CommandField ShowSelectButton="True" />
                                <asp:BoundField DataField="ContactName" HeaderText="ContactName"
                                    />
                                <asp:BoundField DataField="CompanyName" HeaderText="CompanyName"
                                    />
                            </Columns>
                        </asp:GridView>
                    </td>
                    <td valign="top">
                        <asp:DetailsView ID="DetailsView1" runat="server"
                            AutoGenerateRows="True" DataKeyNames="CustomerID"
                            DataSourceID="Details" Height="50px" Width="301px">
                        </asp:DetailsView>
                    </td>
                </tr>
            </table>
        </div>
    </form>
</body>
</html>
```

```

        <asp:SqlDataSource ID="Details" runat="server"
            ConnectionString="<%$ ConnectionStrings:NorthwindConnectionString %>"
            SelectCommand="SELECT * FROM [Customers] WHERE ([CustomerID] =
@CustomerID)">
            <SelectParameters>
                <asp:ControlParameter ControlID="GridView1" Name="CustomerID"
                    PropertyName="SelectedValue"
                    Type="String" />
            </SelectParameters>
        </asp:SqlDataSource>
        <asp:SqlDataSource ID="Customers" runat="server"
            ConnectionString="<%$ ConnectionStrings:NorthwindConnectionString %>"
            SelectCommand="SELECT [CompanyName], [ContactName], [CustomerID] FROM
[Customers]">
        </asp:SqlDataSource>
    </div>
</form>
</body>
</html>

```

The following code example demonstrates how to use the [DetailsView](#) control to add, delete, and edit records.

ASP.NET (C#)Copy

```

<%@ 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">
    void CustomerDetail_ItemInserted(object sender,
        DetailsViewInsertedEventArgs e)
    {
        // Refresh the GridView control after a new record is inserted
        // in the DetailsView control.
        CustomersView.DataBind();
    }

    void CustomerDetail_ItemInserting(object sender,
        DetailsViewInsertEventArgs e)
    {
        // Iterate though the values entered by the user and HTML encode
        // the values. This helps prevent malicious values from being
        // stored in the data source.
        for (int i = 0; i < e.Values.Count; i++)
        {
            if (e.Values[i] != null)
            {
                e.Values[i] = Server.HtmlEncode(e.Values[i].ToString());
            }
        }
    }

    void CustomerDetail_ItemUpdated(object sender,
        DetailsViewUpdatedEventArgs e)
    {
        // Refresh the GridView control after a new record is updated
    }
}

```



```

        // in the DetailsView control.
        CustomersView.DataBind();
    }

    void CustomerDetail_ItemUpdating(object sender,
        DetailsViewUpdateEventArgs e)
    {
        // Iterate though the values entered by the user and HTML encode
        // the values. This helps prevent malicious values from being
        // stored in the data source.
        for (int i = 0; i < e.NewValues.Count; i++)
        {
            if (e.NewValues[i] != null)
            {
                e.NewValues[i] = Server.HtmlEncode(e.NewValues[i].ToString());
            }
        }
    }

    void CustomerDetail_ItemDeleted(object sender,
        DetailsViewDeletedEventArgs e)
    {
        // Refresh the GridView control after a new record is updated
        // in the DetailsView control.
        CustomersView.DataBind();
    }
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>
        DetailsView Example</title>
</head>
<body>
    <form id="Form1" runat="server">
        <h3>
            DetailsView Example</h3>
        <table cellpadding="10">
            <tr>
                <td>
                    <!-- Use a GridView control in combination with -->
                    <!-- a DetailsView control to display master-detail -->
                    <!-- information. When the user selects a store from -->
                    <!-- GridView control, the customers//s detailed -->
                    <!-- information is displayed in the DetailsView -->
                    <!-- control. -->
                    <asp:GridView ID="CustomersView" DataSourceID="Customers"
                        AutoGenerateColumns="False"
                        DataKeyNames="CustomerID" runat="server">
                        <HeaderStyle BackColor="Blue" ForeColor="White" />
                        <Columns>
                            <asp:CommandField ShowSelectButton="True" />
                            <asp:BoundField DataField="ContactName"
                                HeaderText="ContactName" />
                            <asp:BoundField DataField="CompanyName"
                                HeaderText="CompanyName" />
                        </Columns>
                    </asp:GridView>
                </td>
            </tr>
        </table>
    </form>

```

```

<td valign="top">
  <asp:DetailsView ID="CustomerDetail"
    DataSourceID="Details" AutoGenerateRows="false"
    AutoGenerateInsertButton="true"
    AutoGenerateEditButton="true"
    AutoGenerateDeleteButton="true"
    EmptyDataText="No records."
    DataKeyNames="CustomerID" GridLines="Both"
    OnItemInserted="CustomerDetail_ItemInserted"
    OnItemInserting="CustomerDetail_ItemInserting"
    OnItemUpdated="CustomerDetail_ItemUpdated"
    OnItemUpdating="CustomerDetail_ItemUpdating"
    OnItemDeleted="CustomerDetail_ItemDeleted"
    runat="server">
    <HeaderStyle BackColor="Navy" ForeColor="White" />
    <RowStyle BackColor="White" />
    <AlternatingRowStyle BackColor="LightGray" />
    <EditRowStyle BackColor="LightCyan" />
    <Fields>
      <asp:BoundField DataField="CustomerID" HeaderText="CustomerID"
ReadOnly="True" />
      <asp:BoundField DataField="ContactName" HeaderText="ContactName" />
      <asp:BoundField DataField="ContactTitle" HeaderText="ContactTitle"
/>

      <asp:BoundField DataField="CompanyName" HeaderText="CompanyName" />
      <asp:BoundField DataField="Address" HeaderText="Address" />
      <asp:BoundField DataField="City" HeaderText="City" />
      <asp:BoundField DataField="Region" HeaderText="Region" />
      <asp:BoundField DataField="PostalCode" HeaderText="PostalCode" />
      <asp:BoundField DataField="Country" HeaderText="Country" />
      <asp:BoundField DataField="Phone" HeaderText="Phone" />
      <asp:BoundField DataField="Fax" HeaderText="Fax" />
    </Fields>
  </asp:DetailsView>
</td>
</tr>
</table>
<!-- This example uses Microsoft SQL Server and connects -->
<!-- to the Northwind sample database. -->
<!-- It is strongly recommended that each data-bound -->
<!-- control uses a separate data source control. -->
<asp:SqlDataSource ID="Customers" runat="server"
  ConnectionString=
    "<%$ ConnectionStrings:NorthwindConnectionString %%"
  SelectCommand="SELECT [CompanyName], [ContactName], [CustomerID]
    FROM [Customers]">
</asp:SqlDataSource>
<!-- Add a filter to the data source control for the -->
<!-- DetailsView control to display the details of the -->
<!-- store selected in the GridView control. -->
<asp:SqlDataSource ID="Details"
  ConnectionString=
    "<%$ ConnectionStrings:NorthwindConnectionString %%"
  runat="server"
  SelectCommand="SELECT * FROM [Customers]
    WHERE ([CustomerID] = @CustomerID)"
  DeleteCommand="DELETE FROM [Customers]
    WHERE [CustomerID] = @CustomerID"
  InsertCommand="INSERT INTO [Customers] ([CustomerID],

```

```

[CompanyName], [ContactName], [ContactTitle], [Address],
[City], [Region], [PostalCode], [Country], [Phone], [Fax])
VALUES (@CustomerID, @CompanyName, @ContactName, @ContactTitle,
@Address, @City, @Region, @PostalCode, @Country, @Phone, @Fax)"
UpdateCommand="UPDATE [Customers] SET [CompanyName] = @CompanyName,
[ContactName] = @ContactName, [ContactTitle] = @ContactTitle,
[Address] = @Address, [City] = @City, [Region] = @Region,
[PostalCode] = @PostalCode, [Country] = @Country,
[Phone] = @Phone, [Fax] = @Fax
WHERE [CustomerID] = @CustomerID">
<SelectParameters>
  <asp:ControlParameter ControlID="CustomersView"
    Name="CustomerID" PropertyName="SelectedValue"
    Type="String" />
</SelectParameters>
<DeleteParameters>
  <asp:Parameter Name="CustomerID" Type="String" />
</DeleteParameters>
<UpdateParameters>
  <asp:Parameter Name="CompanyName" Type="String" />
  <asp:Parameter Name="ContactName" Type="String" />
  <asp:Parameter Name="ContactTitle" Type="String" />
  <asp:Parameter Name="Address" Type="String" />
  <asp:Parameter Name="City" Type="String" />
  <asp:Parameter Name="Region" Type="String" />
  <asp:Parameter Name="PostalCode" Type="String" />
  <asp:Parameter Name="Country" Type="String" />
  <asp:Parameter Name="Phone" Type="String" />
  <asp:Parameter Name="Fax" Type="String" />
  <asp:Parameter Name="CustomerID" Type="String" />
</UpdateParameters>
<InsertParameters>
  <asp:Parameter Name="CustomerID" Type="String" />
  <asp:Parameter Name="CompanyName" Type="String" />
  <asp:Parameter Name="ContactName" Type="String" />
  <asp:Parameter Name="ContactTitle" Type="String" />
  <asp:Parameter Name="Address" Type="String" />
  <asp:Parameter Name="City" Type="String" />
  <asp:Parameter Name="Region" Type="String" />
  <asp:Parameter Name="PostalCode" Type="String" />
  <asp:Parameter Name="Country" Type="String" />
  <asp:Parameter Name="Phone" Type="String" />
  <asp:Parameter Name="Fax" Type="String" />
</InsertParameters>
</asp:SqlDataSource>
</form>
</body>
</html>

```

The following code example demonstrates how to declaratively add row fields to the [DetailsView](#) control.

ASP.NET (C#)Copy

```

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >

```

```

<head runat="server">
    <title>ASP.NET Example</title>
</head>
<body>
    <form id="Form1" runat="server">
        <table cellpadding="10">
            <tr>
                <td>
                    <!-- Use a GridView control in combination with -->
                    <!-- a DetailsView control to display master-detail -->
                    <!-- information. When the user selects a store from -->
                    <!-- GridView control, the store's detailed -->
                    <!-- information is displayed in the DetailsView -->
                    <!-- control. -->
                    <asp:GridView ID="GridView1" runat="server"
                        DataSourceID="Customers" AutoGenerateColumns="False"
                        DataKeyNames="CustomerID">
                        <Columns>
                            <asp:CommandField ShowSelectButton="True" />
                            <asp:BoundField DataField="ContactName" HeaderText="ContactName" />
                            <asp:BoundField DataField="CompanyName" HeaderText="CompanyName" />
                        </Columns>
                    </asp:GridView>
                </td>
                <td valign="top">
                    <asp:DetailsView ID="DetailsView" runat="server"
                        DataSourceID="Details" AutoGenerateRows="false"
                        DataKeyNames="CustomerID" >
                        <HeaderStyle BackColor="Navy" ForeColor="White" />
                        <Fields>
                            <asp:BoundField DataField="CustomerID" HeaderText="CustomerID"
                                ReadOnly="True" />
                            <asp:BoundField DataField="ContactName" HeaderText="ContactName" />
                            <asp:BoundField DataField="ContactTitle" HeaderText="ContactTitle"
                                />
                            <asp:BoundField DataField="CompanyName" HeaderText="CompanyName" />
                            <asp:BoundField DataField="City" HeaderText="City" />
                            <asp:BoundField DataField="Region" HeaderText="Region" />
                            <asp:BoundField DataField="PostalCode" HeaderText="PostalCode" />
                            <asp:BoundField DataField="Country" HeaderText="Country" />
                        </Fields>
                    </asp:DetailsView>
                </td>
            </tr>
        </table>
        <!-- This example uses Microsoft SQL Server and connects -->
        <!-- to the Northwind sample database. -->
        <!-- It is strongly recommended that each data-bound -->
        <!-- control uses a separate data source control. -->
        <asp:SqlDataSource ID="Customers" runat="server"
            ConnectionString="<%$ ConnectionStrings:NorthwindConnectionString %>"
            SelectCommand="SELECT [CompanyName], [ContactName], [CustomerID] FROM
[Customers]">
        </asp:SqlDataSource>
        <!-- Add a filter to the data source control for the -->
        <!-- DetailsView control to display the details of the -->
        <!-- store selected in the GridView control. -->
        <asp:SqlDataSource ID="Details" runat="server"
            ConnectionString="<%$ ConnectionStrings:NorthwindConnectionString %>"

```

```

        SelectCommand="SELECT * FROM [Customers] WHERE ([CustomerID] =
@CustomerID)">
        <SelectParameters>
            <asp:ControlParameter ControlID="GridView1" Name="CustomerID"
                PropertyName="SelectedValue"
                Type="String" />
        </SelectParameters>
    </asp:SqlDataSource>
</form>
</body>
</html>

```

Remarks

Introduction

The [DetailsView](#) control is used to display a single record from a data source in a table, where each field of the record is displayed in a row of the table. It can be used in combination with a [GridView](#) control for master-detail scenarios.

The [DetailsView](#) control supports the following features:

- Binding to data source controls, such as [SqlDataSource](#).
- Built-in inserting capabilities.
- Built-in updating and deleting capabilities.
- Built-in paging capabilities.
- Programmatic access to the [DetailsView](#) object model to dynamically set properties, handle events, and so on.
- Customizable appearance through themes and styles.

FormView Class

Definition

Namespace:

[System.Web.UI.WebControls](#)

Assembly:

System.Web.dll

Displays the values of a single record from a data source using user-defined templates. The [FormView](#) control allows you to edit, delete, and insert records.

C#Copy

```

[System.Web.UI.ControlValueProperty("SelectedValue")]
public class FormView : System.Web.UI.WebControls.CompositeDataBoundControl,
    System.Web.UI.IDataItemContainer, System.Web.UI.IPostBackEventHandler,
    System.Web.UI.WebControls.IDataBoundItemControl,
    System.Web.UI.WebControls.IPostBackContainer

```

Inheritance

[Object](#)

[Control](#)

[WebControl](#)

[BaseDataBoundControl](#)

[DataBoundControl](#)

[CompositeDataBoundControl](#)

FormView

Attributes

[ControlValuePropertyAttribute](#)

Implements

[IDataItemContainer](#) [INamingContainer](#) [IPostBackEventHandler](#) [IPostBackContainer](#) [IDataBoundControl](#) [IDataBoundItemControl](#)

Examples

The following example demonstrates how to use a [FormView](#) control to display the values from a [SqlDataSource](#) control.

ASP.NET (C#)Copy

```
<%@ Page language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
    <head runat="server">
        <title>FormView Example</title>
    </head>
    <body>
        <form id="form1" runat="server">

            <h3>FormView Example</h3>

            <asp:formview id="EmployeeFormView"
                datasourceid="EmployeeSource"
                allowpaging="true"
                datakeynames="EmployeeID"
                runat="server">

                <itemtemplate>

                    <table>
                        <tr>
                            <td>
                                <asp:image id="EmployeeImage"
                                    imageurl='<%= Eval("PhotoPath") %>'
                                    alternatetext='<%= Eval("LastName") %>'
                                    runat="server"/>
                            </td>
                            <td>
```

```

        <h3><%# Eval("FirstName") %> <%# Eval("LastName") %></h3>
        <%# Eval("Title") %>
    </td>
</tr>
</table>

</itemtemplate>

<pagersettings position="Bottom"
    mode="NextPrevious"/>

</asp:formview>

<!-- This example uses Microsoft SQL Server and connects -->
<!-- to the Northwind sample database. Use an ASP.NET -->
<!-- expression to retrieve the connection string value -->
<!-- from the Web.config file. -->
<asp:sqldatasource id="EmployeeSource"
    selectcommand="Select [EmployeeID], [LastName], [FirstName], [Title],
[PhotoPath] From [Employees]"
    connectionstring="<%$ ConnectionStrings:NorthWindConnectionString%"
    runat="server"/>

</form>
</body>
</html>

```

The following example demonstrates how to use a [FormView](#) control to edit existing records.

Important

The control in this example has a text box that accepts user input, which is a potential security threat. By default, ASP.NET Web pages validate that user input does not include script or HTML elements. For more information, see [Script Exploits Overview](#).

ASP.NET (C#)Copy

```

<%@ 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">

    void EmployeeFormView_ItemUpdating(Object sender, FormViewUpdateEventArgs e)
    {

        // Validate the field values entered by the user. This
        // example determines whether the user left any fields
        // empty. Use the NewValues property to access the new
        // values entered by the user.
        ArrayList emptyFieldList = ValidateFields(e.NewValues);

        if (emptyFieldList.Count > 0)

```

```

{
    // The user left some fields empty. Display an error message.

    // Use the Keys property to retrieve the key field value.
    String keyValue = e.Keys["EmployeeID"].ToString();

    MessageLabel.Text = "You must enter a value for each field of record " +
        keyValue + "<br/>The following fields are missing:<br/><br/>";

    // Display the missing fields.
    foreach (String value in emptyFieldList)
    {
        // Use the OldValues property to access the original value
        // of a field.
        MessageLabel.Text += value + " - Original Value = " +
            e.OldValues[value].ToString() + "<br />";
    }

    // Cancel the update operation.
    e.Cancel = true;
}
else
{
    // The field values passed validation. Clear the
    // error message label.
    MessageLabel.Text = "";
}
}

ArrayList ValidateFields(IOrderedDictionary list)
{
    // Create an ArrayList object to store the
    // names of any empty fields.
    ArrayList emptyFieldList = new ArrayList();

    // Iterate though the field values entered by
    // the user and check for an empty field. Empty
    // fields contain a null value.
    foreach (DictionaryEntry entry in list)
    {
        if (entry.Value == String.Empty)
        {
            // Add the field name to the ArrayList object.
            emptyFieldList.Add(entry.Key.ToString());
        }
    }

    return emptyFieldList;
}

void EmployeeFormView_ModeChanging(Object sender, FormViewModeEventArgs e)
{
    if (e.CancelingEdit)
    {
        // The user canceled the update operation.
    }
}

```



```
// Clear the error message label.
MessageLabel.Text = "";
}
}

</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
  <head runat="server">
    <title>FormView Example</title>
  </head>
  <body>
    <form id="form1" runat="server">

      <h3>FormView Example</h3>

      <asp:formview id="EmployeeFormView"
        datasourceid="EmployeeSource"
        allowpaging="true"
        datakeynames="EmployeeID"
        headertext="Employee Record"
        emptydatatext="No employees found."
        onitemupdating="EmployeeFormView_ItemUpdating"
        onmodechanging="EmployeeFormView_ModeChanging"
        runat="server">

        <headerstyle bgcolor="CornFlowerBlue"
          forecolor="White"
          font-size="14"
          horizontalalign="Center"
          wrap="false"/>
        <rowstyle bgcolor="LightBlue"
          wrap="false"/>
        <pagerstyle bgcolor="CornFlowerBlue"/>

        <itemtemplate>
          <table>
            <tr>
              <td rowspan="6">
                <asp:image id="EmployeeImage"
                  imageUrl='<%# Eval("PhotoPath") %>'
                  alternatetext='<%# Eval("LastName") %>'
                  runat="server"/>
              </td>
              <td colspan="2">

                </td>
            </tr>
            <tr>
              <td>
                <b>Name:</b>
              </td>
              <td>
                <%# Eval("FirstName") %> <%# Eval("LastName") %>
              </td>
            </tr>
            <tr>
              <td>
                <b>Title:</b>
              </td>
              <td>
                <%# Eval("Title") %>
              </td>
            </tr>
            <tr>
              <td>
                <b>Address:</b>
              </td>
              <td>
                <%# Eval("Address") %>
              </td>
            </tr>
            <tr>
              <td>
                <b>Phone:</b>
              </td>
              <td>
                <%# Eval("Phone") %>
              </td>
            </tr>
            <tr>
              <td>
                <b>Email:</b>
              </td>
              <td>
                <%# Eval("Email") %>
              </td>
            </tr>
          </table>
        </itemtemplate>
      </asp:formview>
    </form>
  </body>
</html>
```

```

        </td>
        <td>
            <%=# Eval("Title") %>
        </td>
    </tr>
    <tr>
        <td>
            <b>Hire Date:</b>
        </td>
        <td>
            <%=# Eval("HireDate","{0:d}") %>
        </td>
    </tr>
    <tr style="height:150; vertical-align:top">
        <td>
            <b>Address:</b>
        </td>
        <td>
            <%=# Eval("Address") %><br/>
            <%=# Eval("City") %> <%=# Eval("Region") %>
            <%=# Eval("PostalCode") %><br/>
            <%=# Eval("Country") %>
        </td>
    </tr>
    <tr>
        <td colspan="2">
            <asp:linkbutton id="Edit"
                text="Edit"
                commandname="Edit"
                runat="server"/>
        </td>
    </tr>
</table>
</itemtemplate>
<edititemtemplate>
    <table>
        <tr>
            <td rowspan="6">
                <asp:image id="EmployeeEditImage"
                    imageurl='<%=# Eval("PhotoPath") %>'
                    alternatetext='<%=# Eval("LastName") %>'
                    runat="server"/>
            </td>
            <td colspan="2">

        </td>
    </tr>
    <tr>
        <td>
            <b>Name:</b>
        </td>
        <td>
            <asp:textbox id="FirstNameUpdateTextBox"
                text='<%=# Bind("FirstName") %>'
                runat="server"/>
            <asp:textbox id="LastNameUpdateTextBox"
                text='<%=# Bind("LastName") %>'
                runat="server"/>
        </td>
    </tr>

```

```

</tr>
<tr>
  <td>
    <b>Title:</b>
  </td>
  <td>
    <asp:textbox id="TitleUpdateTextBox"
      text='<%# Bind("Title") %>'
      runat="server"/>
  </td>
</tr>
<tr>
  <td>
    <b>Hire Date:</b>
  </td>
  <td>
    <asp:textbox id="HireDateUpdateTextBox"
      text='<%# Bind("HireDate", "{0:d}") %>'
      runat="server"/>
  </td>
</tr>
<tr style="height:150; vertical-align:top">
  <td>
    <b>Address:</b>
  </td>
  <td>
    <asp:textbox id="AddressUpdateTextBox"
      text='<%# Bind("Address") %>'
      runat="server"/>
    <br/>
    <asp:textbox id="CityUpdateTextBox"
      text='<%# Bind("City") %>'
      runat="server"/>
    <asp:textbox id="RegionUpdateTextBox"
      text='<%# Bind("Region") %>'
      width="40"
      runat="server"/>
    <asp:textbox id="PostalCodeUpdateTextBox"
      text='<%# Bind("PostalCode") %>'
      width="60"
      runat="server"/>
    <br/>
    <asp:textbox id="CountryUpdateTextBox"
      text='<%# Bind("Country") %>'
      runat="server"/>
  </td>
</tr>
<tr>
  <td colspan="2">
    <asp:linkbutton id="UpdateButton"
      text="Update"
      commandname="Update"
      runat="server"/>
    <asp:linkbutton id="CancelButton"
      text="Cancel"
      commandname="Cancel"
      runat="server"/>
  </td>
</tr>

```

```

        </table>
    </edititemtemplate>

    <pagersettings position="Bottom"
        mode="Numeric"/>

</asp:formview>

<br/><br/>

<asp:label id="MessageLabel"
    forecolor="Red"
    runat="server"/>

<!-- This example uses Microsoft SQL Server and connects -->
<!-- to the Northwind sample database. Use an ASP.NET -->
<!-- expression to retrieve the connection string value -->
<!-- from the Web.config file. -->
<asp:sqldatasource id="EmployeeSource"
    selectcommand="Select [EmployeeID], [LastName], [FirstName], [Title],
[Address], [City], [Region], [PostalCode], [Country], [HireDate], [PhotoPath] From
[Employees]"
    updatecommand="Update [Employees] Set [LastName]=@LastName,
[FirstName]=@FirstName, [Title]=@Title, [Address]=@Address, [City]=@City,
[Region]=@Region, [PostalCode]=@PostalCode, [Country]=@Country Where
[EmployeeID]=@EmployeeID"
    connectionstring="<%= $ConnectionStrings:NorthWindConnectionString %>"
    runat="server"/>

</form>
</body>
</html>

```

The following example demonstrates how to use a [FormView](#) control to insert new records.

Important

The control in this example has a text box that accepts user input, which is a potential security threat. By default, ASP.NET Web pages validate that user input does not include script or HTML elements. For more information, see [Script Exploits Overview](#).

ASP.NET (C#)Copy

```

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
    <head runat="server">
        <title>FormView InsertItemTemplate Example</title>
    </head>
    <body>
        <form id="form1" runat="server">

```

<h3>FormView InsertItemTemplate Example</h3>

```
<asp:formview id="EmployeeFormView"
  datasourceid="EmployeeSource"
  allowpaging="true"
  datakeynames="EmployeeID"
  emptydatatext="No employees found."
  runat="server">

  <rowstyle bgcolor="LightGreen"
    wrap="false"/>
  <insertrowstyle bgcolor="LightBlue"
    wrap="false"/>

  <itemtemplate>
    <table>
      <tr>
        <td rowspan="5">
          <asp:image id="CompanyLogoImage"
            imageurl="~/Images/Logo.jpg"
            alternatetext="Company logo"
            runat="server"/>
        </td>
        <td colspan="2">

          </td>
        </tr>
        <tr>
          <td>
            <b>Name:</b>
          </td>
          <td>
            <%# Eval("FirstName") %> <%# Eval("LastName") %>
          </td>
        </tr>
        <tr>
          <td>
            <b>Title:</b>
          </td>
          <td>
            <%# Eval("Title") %>
          </td>
        </tr>
        <tr>
          <td colspan="2">
            <asp:linkbutton id="NewButton"
              text="New"
              commandname="New"
              runat="server"/>
          </td>
        </tr>
      </table>
    </itemtemplate>
    <insertitemtemplate>
      <table>
        <tr>
          <td rowspan="4">
            <asp:image id="CompanyLogoEditImage"
```

```

        imageurl="~/Images/Logo.jpg"
        alternatetext="Company logo"
        runat="server"/>
    </td>
    <td colspan="2">

    </td>
</tr>
<tr>
    <td>
        <b><asp:Label runat="server"
            AssociatedControlID="FirstNameInsertTextBox"
            Text="Name" />:</b>
    </td>
    <td>
        <asp:textbox id="FirstNameInsertTextBox"
            text='<%# Bind("FirstName") %>'
            runat="server"/>
        <asp:textbox id="LastNameInsertTextBox"
            text='<%# Bind("LastName") %>'
            runat="server"/>
    </td>
</tr>
<tr>
    <td>
        <b><asp:Label runat="server"
            AssociatedControlID="TitleInsertTextBox"
            Text="Title" />:</b>
    </td>
    <td>
        <asp:textbox id="TitleInsertTextBox"
            text='<%# Bind("Title") %>'
            runat="server"/>
    </td>
</tr>
<tr>
    <td colspan="2">
        <asp:linkbutton id="InsertButton"
            text="Insert"
            commandname="Insert"
            runat="server" />
        <asp:linkbutton id="CancelButton"
            text="Cancel"
            commandname="Cancel"
            runat="server" />
    </td>
</tr>
</table>
</insertitemtemplate>

</asp:formview>

<!-- This example uses Microsoft SQL Server and connects -->
<!-- to the Northwind sample database. Use an ASP.NET -->
<!-- expression to retrieve the connection string value -->
<!-- from the Web.config file. -->
<asp:sqldatasource id="EmployeeSource"
    selectcommand="Select [EmployeeID], [LastName], [FirstName], [Title],
[PhotoPath] From [Employees]"

```

```
insertcommand="Insert Into [Employees] ([LastName], [FirstName], [Title])
VALUES (@LastName, @FirstName, @Title)"
connectionstring="<%$ ConnectionStrings:NorthWindConnectionString%"
runat="server"/>

</form>
</body>
</html>
```

Remarks

:

Introduction

The [FormView](#) control is used to display a single record from a data source. It is similar to the [DetailsView](#) control, except it displays user-defined templates instead of row fields. Creating your own templates gives you greater flexibility in controlling how the data is displayed. The [FormView](#) control supports the following features:

- Binding to data source controls, such as [SqlDataSource](#) and [ObjectDataSource](#).
- Built-in inserting capabilities.
- Built-in updating and deleting capabilities.
- Built-in paging capabilities.
- Programmatic access to the [FormView](#) object model to dynamically set properties, handle events, and so on.
- Customizable appearance through user-defined templates, themes, and styles.

Repeater Class

Definition

Namespace:

[System.Web.UI.WebControls](#)

Assembly:

System.Web.dll

A data-bound list control that allows custom layout by repeating a specified template for each item displayed in the list.

C#Copy

```
public class Repeater : System.Web.UI.Control, System.Web.UI.INamingContainer
```

Inheritance

[Object](#)

[Control](#)

Repeater
Derived
 [System.Web.DynamicData.FilterRepeater](#)
Implements
 [INamingContainer](#)

Examples

A Visual Studio Web site project with source code is available to accompany this topic: [Download](#).

The following code example demonstrates how to use two simple [Repeater](#) controls on a page. The [DataSource](#) property is used to specify the data source for the [Repeater](#) control. The first [Repeater](#) displays its items in a table; the second [Repeater](#) displays its items in a comma-separated list.

ASP.NET (C#)Copy

```
<%@ Page Language="C#" AutoEventWireup="True" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
  <head>
    <title>Repeater Example</title>
  <script language="C#" runat="server">
    void Page_Load(Object Sender, EventArgs e) {
      if (!IsPostBack) {
        ArrayList values = new ArrayList();

        values.Add(new PositionData("Microsoft", "Msft"));
        values.Add(new PositionData("Intel", "Intc"));
        values.Add(new PositionData("Dell", "Dell"));

        Repeater1.DataSource = values;
        Repeater1.DataBind();

        Repeater2.DataSource = values;
        Repeater2.DataBind();
      }
    }

    public class PositionData {

      private string name;
      private string ticker;

      public PositionData(string name, string ticker) {
        this.name = name;
        this.ticker = ticker;
      }

      public string Name {
        get {
```



```

        return name;
    }
}

public string Ticker {
    get {
        return ticker;
    }
}
}

</script>

</head>
<body>

<h3>Repeater Example</h3>

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

    <b>Repeater1:</b>

    <br />

    <asp:Repeater id="Repeater1" runat="server">
        <HeaderTemplate>
            <table border="1">
                <tr>
                    <td><b>Company</b></td>
                    <td><b>Symbol</b></td>
                </tr>
            </table>
        </HeaderTemplate>

        <ItemTemplate>
            <tr>
                <td> <%=# DataBinder.Eval(Container.DataItem, "Name") %> </td>
                <td> <%=# DataBinder.Eval(Container.DataItem, "Ticker") %> </td>
            </tr>
        </ItemTemplate>

        <FooterTemplate>
            </table>
        </FooterTemplate>

    </asp:Repeater>
    <br />

    <b>Repeater2:</b>
    <br />
    <asp:Repeater id="Repeater2" runat="server">

        <HeaderTemplate>
            Company data:
        </HeaderTemplate>

        <ItemTemplate>
            <%=# DataBinder.Eval(Container.DataItem, "Name") %> (<%=#
DataBinder.Eval(Container.DataItem, "Ticker") %>)
        </ItemTemplate>
    </asp:Repeater>

```

```

        <SeparatorTemplate>, </SeparatorTemplate>
    </asp:Repeater>
</form>
</body>
</html>

```

The following code example demonstrates how to use the [DataSourceID](#) property to specify the data source for a [Repeater](#) control. The [DataSourceID](#) property is set to the [ID](#) property of the [SqlDataSource](#) control used to retrieve the data. When the page is loaded, the [Repeater](#) control automatically binds to the data source specified by the [SqlDataSource](#) control and the data is displayed to the user.

ASP.NET (C#)Copy

```

<%@ page language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
    <head>
        <title>Repeater.DataSourceID Property Example</title>
    </head>

    <body>
        <form id="Form1" runat="server">

            <h3>Repeater.DataSourceID Property Example</h3>

            <asp:repeater id="Repeater1"
                datasourceid="SqlDataSource1"
                runat="server">

                <headertemplate>
                    <table border="1">
                        <tr>
                            <td><b>Product ID</b></td>
                            <td><b>Product Name</b></td>
                        </tr>
                    </table>
                </headertemplate>

                <itemtemplate>
                    <tr>
                        <td> <%# Eval("ProductID") %> </td>
                        <td> <%# Eval("ProductName") %> </td>
                    </tr>
                </itemtemplate>

                <footertemplate>
                    </table>
                </footertemplate>
            </asp:repeater>

            <asp:sqldatasource id="SqlDataSource1"
                connectionstring="<%$ ConnectionStrings:NorthWindConnection%>"
                selectcommand="SELECT ProductID, ProductName FROM [Products] Where
ProductID <= 10"
                runat="server">

```

```
</asp:sqldatasource>  
  
</form>  
</body>  
</html>
```

Remarks

In this topic:

Introduction

The [Repeater](#) control is a basic templated data-bound list. It has no built-in layout or styles, so you must explicitly declare all layout, formatting, and style tags within the control's templates.

The [Repeater](#) control allows you to split markup tags across the templates. To create a table using templates, include the begin table tag (<table>) in the [HeaderTemplate](#), a single table row tag (<tr>) in the [ItemTemplate](#), and the end table tag (</table>) in the [FooterTemplate](#).

The [Repeater](#) control has no built-in selection capabilities or editing support. You can use the [ItemCommand](#) event to process control events that are raised from the templates to the control.

What is ADO.Net?

We create two-tier models that bridge ASP.NET with backend databases, enabling applications to access diverse types of data using the same methodology and to connect to a SQL Server database using a different set of classes. In ADO.net architecture, we use a two-tier model to create a bridge between ASP.net and the backend database, through which applications can access various types of data using the same methodology and connect to a SQL Server database using a different set of classes. We can keep connections with the database and access data using the connected model, and we can access data using disconnected objects with the other model, providing two different connections. The Microsoft .net framework is also included, and it is a set of classes that provides the foundation for .NET.

How Does ADO.NET Work?

1. In connected mode, data is read in forward-only read-only mode and data is updated, deleted, inserted, and selected in disconnected mode, in which data is read and updated in both read and update modes.
2. In connected mode, a single table can be held, but when there are multiple tables to be held, they must be held separately.
3. In connected mode, objects are forward-only-read-only; while in disconnected mode, they can process data in any dimension.

Examples:

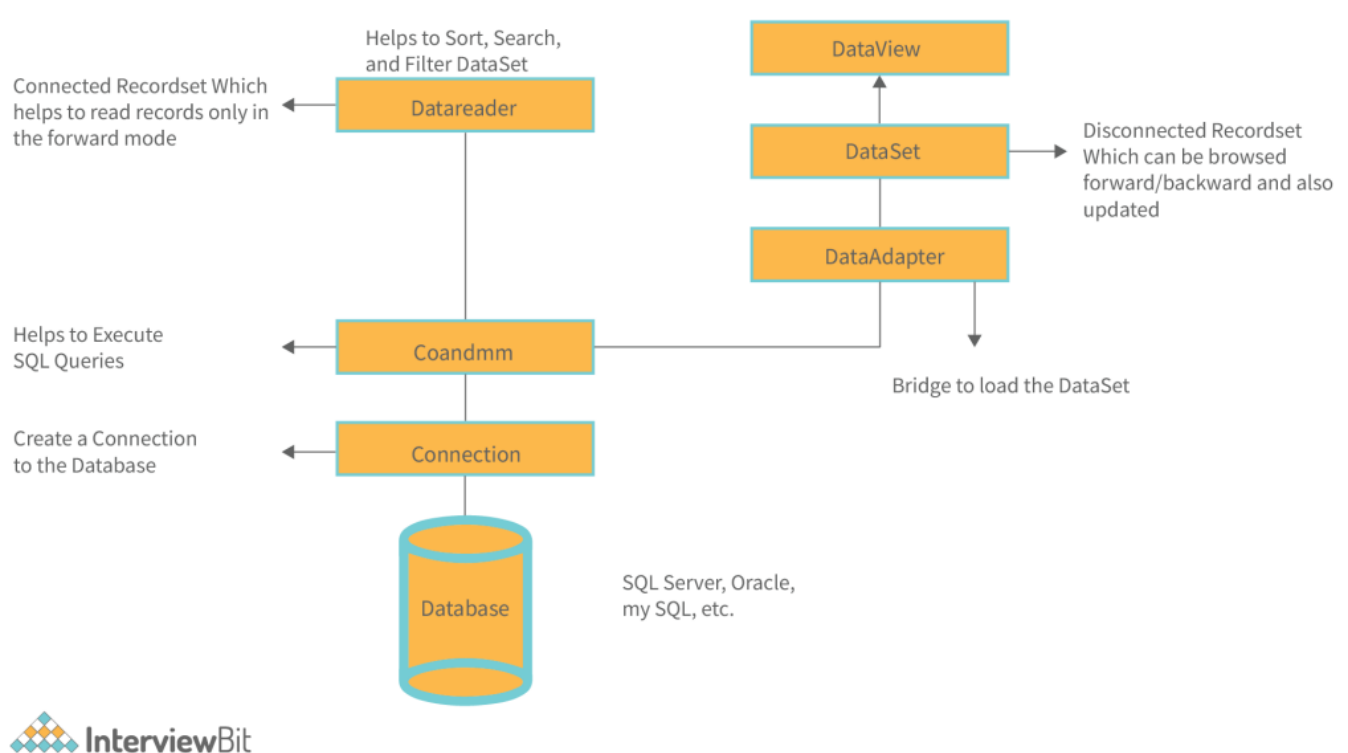
1. When data is read from a data reader, it maintains the connection open until all records have been retrieved.
2. When a DataSet receives all its records at once and closes the connection, it is still available to data sources that are connected in a disconnected architecture

ADO.NET Architecture

The ADO.NET architecture comprises six important components. They are as follows:

1. Connection
2. Command
3. DataReader
4. DataAdapter
5. DataSet
6. DataView

The command and connection objects shown in the preceding diagram are two of the four components that must be present in a BDC command object. A BDC command object must include one of these two components. A connection object is needed regardless of what operations are performed on it, including Insert, Update, Delete, and Select. In the preceding diagram, have a look at the image.



Let's take a look at each of the parts individually:

Connection: The connection object is the first important component of your application. It is required to connect to a backend database that may be SQL Server, Oracle, MySQL, etc. You must have two things to create a connection object. Your database Machine name or IP address or someplace where it is stored is where it is. The second thing is security credentials, such as whether it's a Windows authentication or username and password-based authentication. The connection is created using the connection object and a backend data source must be connected to using the connection.

Command: The second important component is the command object. When we discuss databases such as SQL Server, Oracle, MySQL, then speak SQL, it is the command object that we use to create SQL queries. After you create your SQL queries, you can execute them over the connection using the command object. You can go either the DataSet or the DataReader way with DTS. In general, you should choose which method you require based on the situation. Note: You can go either the DataSet or the DataReader way with DTS.

DataReader: We can only read the records in the forward mode with DataReader. Here, you should familiarize yourself with three things: read-only, connected, and forward modes.

DataSet: A disconnected recordset can be browsed in both directions, and it is also possible to insert, update, or delete data sets. The DataAdapter fills a DataSet using data.

DataAdapter: The DataAdapter performs an operation on the data from the command object and then writes the data set to the dataset.

DataView Class: A DataView enables you to modify the appearance of the data stored in a DataTable, a data-binding skill that is frequently employed in data-view applications. You may alter the sort order of data in a table or filter it based on row state or on a filter expression using a DataView.

Features of ADO.NET

Features of ADO.NET include:

1. When viewed as text-based formats, XML documents are obviously negotiable. ADO.NET exchanges data using XML, regardless of its complexity, and for internal purposes.
2. We can model our application in separate layers, which is what ADO.NET is built around.
3. A programming style in which words are used to construct assertions or evaluate expressions is called word-based programming. The following code fragment illustrates how to select the "Ranks" column from "Scaler" in the "Student" table using word-based programming:
`DataSet.Student("Scaler").Ranks;`
4. The data architecture is simple to scale as it involves only disconnected data on the server. Because everything is handled on the client-side, performance is improved.
5. The growing number of clients requiring degraded performance as it uses disconnected data access is accommodated by the application's use of lock connections that last longer. In addition, the application can afford to make the programmers conserve resources and allow users to access data simultaneously.

Connected and Disconnected architecture in ADO.NET

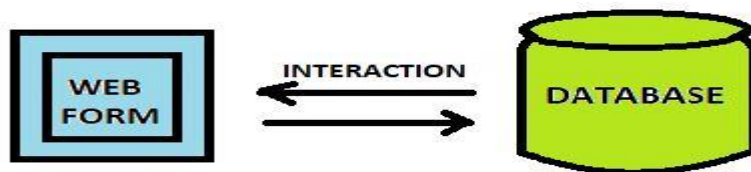
ADO.NET is a data access technology that allows interaction between applications and databases. The ADO.NET framework supports two models of data access architecture i.e. Connection Oriented Data Access Architecture and Disconnected Data Access Architecture. In this article, we'll study about these two modes of architecture.

The architecture followed by ADO.NET for the connectivity with the database is categorized into two modes:-

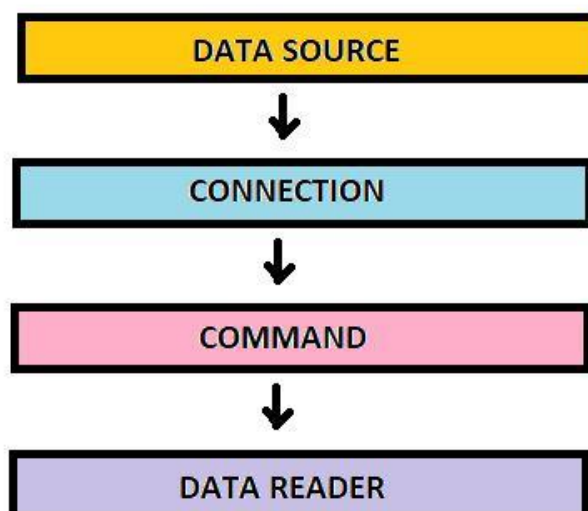
Types of Architecture

ADO.NET is both connection-oriented as well as disconnection oriented. Depending upon the functionality of an application, we can make it connection-oriented or disconnection oriented. We can even use both the modes together in a single application.

Connected Architecture



DataReader in Connected architecture



In order to make an object of the DataReader class, we never use the new keyword instead we call the ExecuteReader() of the command object. For e.g.

```
SqlCommand cmd= new SqlCommand("Select * from Table");  
  
SqlDataReader rdr=cmd.ExecuteReader(cmd);
```

Here cmd.ExecuteReader() executes the command and creates the instance of DataReader class and loads the instance with data. Therefore, we do not use the 'new' keyword.

2. Disconnected Architecture

DataAdapter in Disconnected architecture finally the modifications are done in the Dataset which is passed to the DataAdapter which updates the database. DataAdapter takes the decision for the establishment and termination of the connection.



DataAdapter is required for connectivity with the database. DataAdapter established a connection with the database and fetches the data from the database and fill it into the DataSet. And finally, when the task is completed it takes the data from the DataSet and updates it into the database by again establishing the connection.

It can be said that DataAdapter acts as a mediator between the application and database which allows the interaction in disconnected architecture.

For example:-

```
public DataTable GetTable(string query)  
{  
  
    SqlDataAdapter adapter = new SqlDataAdapter(query, ConnectionString);  
    DataTable Empl = new DataTable();  
    adapter.Fill(Empl);  
    return Empl;  
}
```



```
}
```

In the above lines of code, the object of the SqlDataAdapter is responsible for establishing the connection. It takes query and ConnectionString as a parameter. The query is issued on the database to fetch the data. ConnectionString allows connectivity with the database. The fill() of the SqlDataAdapter class adds the Table.

Crystal Reports

Crystal Reports is a popular Windows-based report writer solution that allows a developer to create reports and dashboards from a variety of data sources with a minimum of code to write. Crystal Reports is owned and developed by [SAP](#).

As a [business intelligence](#) application for individual users or small and medium-size businesses ([SMBs](#)), SAP Crystal Reports 2020 is designed to work with databases to help users analyze and interpret important information. While users can create simple reports, the software also offers comprehensive tools needed to produce complex or specialized reports.

Crystal Reports is designed to produce reports from virtually any data source. Formulas, cross-tabs, sub-reports and conditional formatting help make sense of data and uncover important relationships that might otherwise be hidden. [Data visualization](#) tools such as geographic maps and graphs communicate information visually to help in understanding data analysis.

Crystal Reports can publish in a variety of formats including Microsoft Word and Excel, e-mail and over the Web. Advanced Web reporting allows all members of in a workgroup view and update shared reports inside a web browser.

Application and web developers can save time and meet their user's needs by integrating the report processing power of Crystal Reports into their applications. Adding reporting to applications is aided by support for .NET and [Java](#).

Top competitors of Crystal Reports include Logi Analytics, Tableau Desktop, Cognos Analytics, Sisense and Chartio.