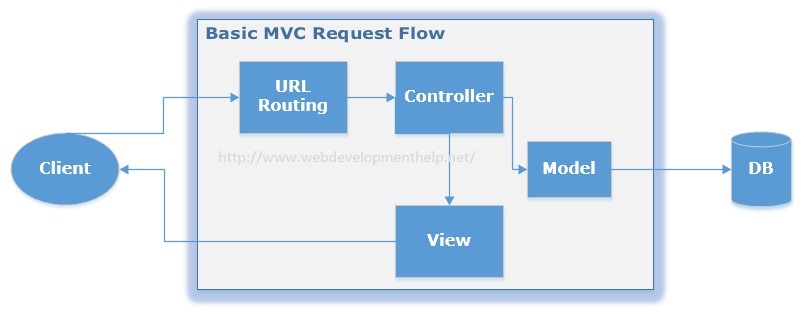
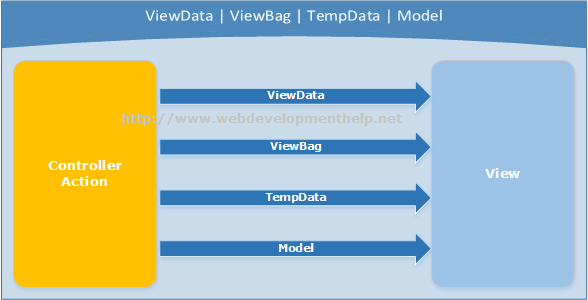
**Passing data in MVC**

**ASP.NET MVC** is a framework that facilitates building web applications based on MVC (Model-View-Controller) design pattern. Request coming from client reaches the *Controller* through URL Rewriting Module. *Controller* decides which model to use in order to fulfill the request. Further passing the *Model* data to *View* which then transforms the *Model* data and renders response to client as shown in following basic level request flow diagram.

[](http://www.webdevelopmenthelp.net/wp-content/uploads/2014/06/MVC-Request-Flow2.jpg)

In this ASP.NET MVC Tutorial, we will discuss and implement different options to pass data from ASP.NET MVC *Controller* to *View*. Following are the different available options to pass data from a *Controller* to *View* in ASP.NET MVC along with Introduction and Background:

* [Introduction and Background](http://www.webdevelopmenthelp.net/2014/06/asp-net-mvc-pass-data-controller-view.html#Introduction)
* [ViewBag](http://www.webdevelopmenthelp.net/2014/06/asp-net-mvc-pass-data-controller-view.html#ViewBag)
* [ViewData](http://www.webdevelopmenthelp.net/2014/06/asp-net-mvc-pass-data-controller-view.html#ViewData)
* [TempData (TempData Vs Session)](http://www.webdevelopmenthelp.net/2014/06/using-tempdata-asp-net-mvc.html)
* [Model](http://www.webdevelopmenthelp.net/2014/06/using-model-pass-data-asp-net-mvc.html)

Let’s discuss all these options step by step in this series.

**Introduction**

If we want to maintain state between a *Controller* and corresponding *View*- ***ViewData*** and ***ViewBag*** are the available options but both of these options are limited to a single server call (meaning it’s value will be null if a redirect occurs). But if we need to maintain state from one *Controller* to another (redirect case), then ***TempData*** is the other available option.

It’s common that initially it might be a bit difficult for a ASP.NET WebForms developer to digest above flow and need for options to pass data from *Controller* to *View.*Because in WebForms approach, Controller and View are tightly coupled to each other. Please follow the link for a detailed comparison of the [differences between ASP.NET WebForms and ASP.NET MVC](http://www.webdevelopmenthelp.net/2013/10/difference-between-asp-net-webform-and-asp-net-mvc.html) here.

For the purpose of implementation, we will take earlier [ASP.NET MVC tutorial](http://www.webdevelopmenthelp.net/2014/02/building-asp-net-mvc5-application.html) on this blog as base and implement with different options. If you haven’t gone through the article, please read “[Building your first ASP.NET MVC application in 4 simple steps](http://www.webdevelopmenthelp.net/2014/02/building-asp-net-mvc5-application.html)” first.

**ViewBag Example**

As we discussed earlier that *ViewBag* and *ViewData* serves the same purpose but *ViewBag* is basically a dynamic property (a new C# 4.0 feature) having advantage that it doesn’t have typecasting and null checks.  
So, In order to pass data from Controller to View using ViewBag, we will modify our EmployeeController code as follows:

*public class EmployeeController : Controller*  
*{*  
*// GET: /Employee/*  
*public ActionResult Index()*  
*{*  
*ViewBag.EmployeeName = “Muhammad Hamza”;*  
*ViewBag.Company = “Web Development Company”;*  
*ViewBag.Address = “Dubai, United Arab Emirates”;*

*return View();*  
*}*  
*}*

And to get Employee details passed from Controller using ViewBag, View code will be as follows:

*<body>*  
*<div>*  
*<h1>Employee (ViewBag Data Example)</h1>*  
*<div>*  
*<b>Employee Name:</b> @ViewBag.EmployeeName<br />*  
*<b>Company Name:</b> @ViewBag.Company<br />*  
*<b>Address:</b> @ViewBag.Address<br />*  
*</div>*  
*</div>*  
*</body>*

In order to see the above changes in action run the solution, we will find the following output.

[](http://www.webdevelopmenthelp.net/wp-content/uploads/2014/06/ViewBag.jpg)

### ViewData Example

As compared to ViewBag, ViewData is a dictionary object which requires typecasting as well as null checks. Same above implementation using ViewData can be achieved as follows:

 public class EmployeeController : Controller  
 {  
           // GET: /Employee/  
          public ActionResult Index()  
         {  
                     ViewData[“EmployeeName”] = “Muhammad Hamza”;  
                     ViewData[“Company”] = “Web Development Company”;  
                     ViewData[“Address”] = “Dubai, United Arab Emirates”;

                     return View();  
         }  
  }

And to get Employee details passed from Controller using ViewBag, View code will be as follows:

  <body>  
    <div>  
          <h1>Employee (ViewBag Data Example)</h1>  
          <div>  
                       <b>Employee Name:</b> @ViewData[“EmployeeName”]<br />  
                      <b>Company Name:</b> @ViewData[“Company”]<br />  
                      <b>Address:</b> @ViewData[“Address”]<br />  
          </div>  
    </div>  
  </body>

Run the application to view the following output.

[](http://www.webdevelopmenthelp.net/wp-content/uploads/2014/06/ViewData.jpg)  
Hopefully, this ASP.NET MVC Tutorial will provide reader with a better understanding of passing data from Controller to View in ASP.NET MVC using ViewBag and ViewData.

### TempData

TempData in ASP.NET MVC is basically a dictionary object derived from TempDataDictionary. TempData stays for a subsequent HTTP Request as opposed to other options (ViewBag and ViewData) those stay only for current request. So, TempdData can be used to maintain data between controller actions as well as redirects.

***Note:*** *Just like ViewData, typecasting and null checks required for TempData also in order to avoid errors.*

Let’s see how we can use TempData in a practical scenario to pass data from one controller action to another.

*//Controller Action 1 (TemporaryEmployee)*

*public ActionResult TemporaryEmployee()*  
*{*  
*Employee employee = new Employee*  
*{*  
*EmpID = “121”,*  
*EmpFirstName = “Imran”,*  
*EmpLastName = “Ghani”*  
*};*

*TempData[“Employee”] = employee;*  
*return RedirectToAction(“PermanentEmployee”);*  
*}*

*//Controller Action 2(PermanentEmployee)*

*public ActionResult PermanentEmployee()*  
*{*  
*Employee employee = TempData[“Employee”] as Employee;*  
*return View(employee);*  
*}*

As in above example, we store an employee object in TempData in Controller Action 1 (i.e. TemporaryEmployee) and retrieve it in another Controller Action 2 (i.e. PermanentEmployee). But If we try to do the same using ViewBag or ViewData, we will get null in Controller Action 2 because only TempData object maintains data between controller actions.

### Model

Now in this part, we are going to implement another approach i.e. using Model for passing data from ASP.NET MVC Controller to View. As we already know that in MVC (Model, View, Controller) pattern, Model represents our domain model corresponding to tables in database. So, we can use this model class to serve the purpose.

Consider the following Model class we have already used in our previous implementations:

*namespace MyMVCApp.Models*  
*{*  
*public class Employee*  
*{*  
*public string EmpID { get; set; }*  
*public string EmpFirstName { get; set; }*  
*public string EmpLastName { get; set; }*  
*}*  
*}*

Now, we are done with our Model class (i.e. Employee). For the purpose of this implementation, we are loading data directly to our Model in EmployeeController. In a real implementation, we must be fetching this data from a data source e.g. a database. So, the Controller has following code:

*public class EmployeeController : Controller*  
*{*  
*public ActionResult Index()*  
*{*  
*List<Employee> employees = new List<Employee>()*  
*{*  
*new Employee{EmpID = “1”, EmpFirstName = “Imran”, EmpLastName = “Ghani”},*  
*new Employee{EmpID = “2”, EmpFirstName = “Rizwan”, EmpLastName = “Mukhtar”},*  
*new Employee{EmpID = “3”, EmpFirstName = “Rehan”, EmpLastName = “Ahmad”},*  
*new Employee{EmpID = “4”, EmpFirstName = “Zeeshan”, EmpLastName = “Khalid”},*  
*new Employee{EmpID = “5”, EmpFirstName = “Sajid”, EmpLastName = “Majeed”}*  
*};*

*return View(employees);*  
*}*  
*}*

In above code example, you can see that as opposite to ViewData and ViewBag, we are passing data (i.e. employees) as parameter to View instead of placing in another store and calling as View().

inally, updating our View to get and display data passed from Controller as:

*@model IEnumerable<MyMVCApp.Models.Employee>*

*<html>*  
*<head>*  
*<meta name=”viewport” content=”width=device-width” />*  
*<title>Employee Index Page</title>*  
*</head>*  
*<body>*  
*<div>*  
*<h1>Employee (Using Model)</h1>*  
*<div>*  
*@foreach(var employee in Model){*  
*<p>@employee.EmpID  @employee.EmpFirstName  @employee.EmpLastName</p>*  
*}*  
*</div>*  
*</div>*  
*</body>*  
*</html>*

At the top of our View (i.e. Index.cshtml), we have referred to our Model class (i.e. Employee) using IEnumerable. Further in View, we are using foreach to access and display data as needed. When we run this application, our View will display the Model data from Controller as follows:

[](http://www.webdevelopmenthelp.net/wp-content/uploads/2014/06/Result.jpg)

Using Model to pass data to View in ASP.NET is pretty simple as well as more appropriate because we are fetching data from permanent store and loading in our Model classes in most of the cases. Also, we use it to perform all CRUD (Create, Retrieve, Update, Delete) operations. Hopefully, this ASP.NET MVC Tutorial will help in understanding the mentioned approach in a more effective way.

**The article is written by Web Development Tutorial**