# Action Result In ASP.NET MVC

* [](https://www.c-sharpcorner.com/members/zain-ul-hassan2)

* [Zain Ul Hassan](https://www.c-sharpcorner.com/members/zain-ul-hassan2)

* Dec 03, 2020

* 481.4k

* 23

* 22

[Download .NET & JAVA Office File API for Free](https://www.c-sharpcorner.com/AdRedirector.aspx?AdId=133&target=https://www.e-iceblue.com/download/free-products.html?aff_id=100)

## Introduction and Background

This article will demonstrate the basic to advanced concepts of action results in ASP.NET MVC. Last week, one of my friends asked the question “Why there is ContentResult data type in the action method and what is its purpose?” I am dedicating this article to him. I hope he will like this.

The topics to be covered are,

* What is ActionResult?
  + Content Returning Results
  + Redirection Results
  + Status Results

**The Question**

The first part of the question is: "Why there is ContentResult data type in the action method?". To understand this, you have to understand the data types in ASP.NET MVC 5 because the ContentResult is a data type whose base data type is ActionResult, hence you have to understand Action Result.

Let’s get started with Action Result.

## Action Result in ASP.NET MVC

Action Result is actually a data type. When it is used with action method, it is called return type. As you know, an action is referred to as a method of the controller, the Action Result is the result of action when it executes. In fact, Action Result is a return type. This return type has many other derived types. First, look at the base and derived types of ActionResult.

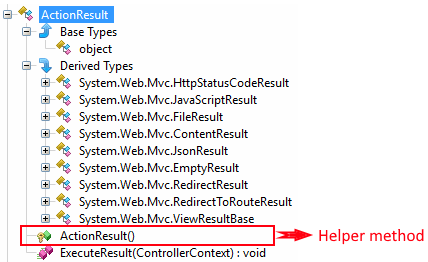
1. **namespace** System.Web.Mvc
2. {
3. **public** **abstract** **class** ActionResult
4. {
5. //
6. // Summary:
7. //     Initializes a new instance of the System.Web.Mvc.ActionResult class.
8. **protected** ActionResult();
9. }
10. }

As we see in the Visual Studio, the ActionResult type is coming from System.Web.Mvc assembly. And the problem is that we can’t see inside this assembly with “Go to Definition” feature of Visual Studio.

To see inside the assembly, you can either follow the following steps after downloading ILSpy or you can watch this video by clicking [here](https://github.com/Syed-Zain-Ul-Hassan/ASP.NET-MVC-5/tree/master/ActionResult%2C%20ActionFilter%20and%20Action%20Selectors).

Steps are as follows.

1. I am giving you the direct link to download the ILSpy. Click [here](https://github.com/Syed-Zain-Ul-Hassan/ASP.NET-MVC-5/raw/master/ActionResult%2C%20ActionFilter%20and%20Action%20Selectors/ILSpy_binaries_3.1.0.3652.zip).
2. If you don’t have the System.Web.Mvc assembly, then you can download after clicking [here](https://github.com/Syed-Zain-Ul-Hassan/ASP.NET-MVC-5/raw/master/ActionResult%2C%20ActionFilter%20and%20Action%20Selectors/System.Web.Mvc.zip).
3. Run ILSpy after extracting it.
4. Click on File (on the top-left-corner)
5. Select “Open” option and open the assembly from where you downloaded it.
6. After selecting, the assembly will load in ILSpy software.
7. Expand the assembly, you will see ActionResult, expand it also, you will see the base and derived types also.
8. The types of ActionResult is shown in the image below:



Now, you can see that the Action Result is a base data type whose derived types are HttpStatusCodeResult, JavaScriptResult, FileResult, ContentResult, JsonResult, EmptyResult, RedirectResult, RedirectToRouteResult, ViewResult. And, there must be a helper method against each data type (either base or derived type).

When we go into ActionResult, we see that it is an abstract class.

1. **namespace** System.Web.Mvc
2. {
3. **public** **abstract** **class** ActionResult
4. {
5. //
6. // Summary:
7. //     Initializes a new instance of the System.Web.Mvc.ActionResult class.
8. **protected** ActionResult();
9. }
10. }

It is an abstract class because actions of controller can return different types of data at the same time. So, it has to be abstract so that every HTTP request could handle properly. See the code below,

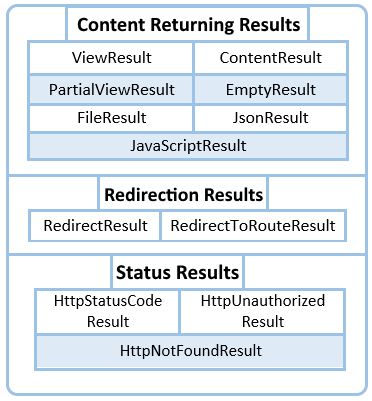
1. **public** ActionResult Index()
2. {
3. **bool** answer = DateTime.Now.Day + 2 == 5;
4. **if** (answer)
5. {
6. **return** Content("Correct");
7. }
8. **else**
9. {
10. **return** View();
11. }
12. }

 As you can see, the same action method “Index” is returning two different types named Content and View; if you want to return multiple types, you have to use base type as ActionResult.

The above concept also answers the question “When to choose base type ActionResult or derived type?”

Choosing the derived type for a specific result is a good practice, but when you want that your action method should return multiple types, then you have to use base type ActionResult.

Now, the important concept comes. There are three categories for the derived types. Let’s take a look at it.



As you can see, there are three categories of data types of ActionResult,

1. Content Returning Results
2. Redirection Results
3. Status Results

Let’s understand each.

## Content Returning Results

As the name depicts these results are used for returning the content to the browser. There are 7 types of content returning results:

* ViewResult
* PartialViewResult
* ContentResult
* EmptyResult
* FileResult
* JsonResult
* JavaScriptResult

### **ViewResult**

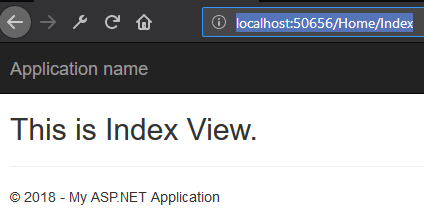
ViewResult is a datatype which is responsible for returning the View. Let’s look at its example.

1. **public** **class** HomeController : Controller
2. {
3. **public** ViewResult Index()
4. {
5. **return** View();
6. }
7. }

And the code of Index view is as follows,

1. <h2>
2. This is Index View.
3. </h2>

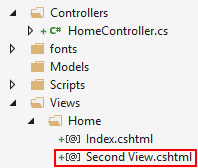
As you can see in the above code, index action method is returning the View using ViewResult type. After this, MVC will find the View having name “Index” from the “Home” folder in the Views folder. And the output is as follows,



The interesting thing with ViewResult type is, you can return a different View regardless of using View having the same name as that of the action method. Look at an example.

1. **public** **class** HomeController : Controller
2. {
3. **public** ViewResult Index()
4. {
5. **return** View(“Second View” );
6. }
7. }

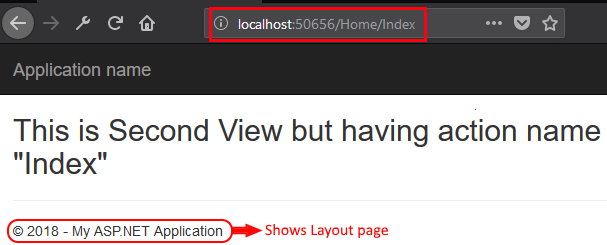
You can see the figure below that we “Second View” page in views folder,



And the code of “Second View” page is as follows,

1. <h2>This is Second View but having action name "Index"</h2>

You can see above, now MVC will find the view having name “Second View” rather than finding the view having the same name as that of the action name. Output is as follows,



Hence, you can call any other view using different action name with the help of ViewResult type.

### **PartialViewResult**

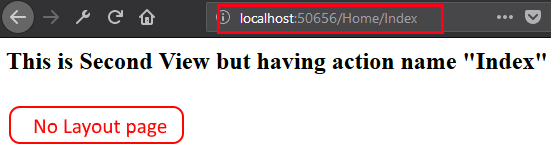
It is the type which is used to return the partial view page rather than returning the regular view page. Look at its example.

1. **public** **class** HomeController : Controller
2. {
3. **public** PartialViewResult Index()
4. {
5. **return** PartialView("Second View");
6. }
7. }

And the code of “Second View” is shown below.

1. <h2>This is Second View but having action name "Index"</h2>

Now the output is different, let’s see below,



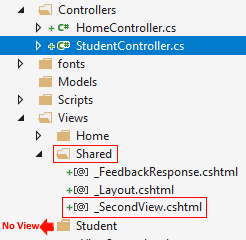
As you can see the output is the same as in the previous example. But the difference is here we can’t see the line “© 2018 - My ASP.NET Application” because it is the line from Layout page, which is absent here due to the usage of PartialViewResult datatype.

As you can see the output of PartialViewResult is same as that of ViewResult. But the content of layout page is missing. It is just an example. This is not the real-time example.

Now, let’s look at the example where we’ll see layout page. Make a new controller named as “Student” and an action method named as “Name”, as shown below,

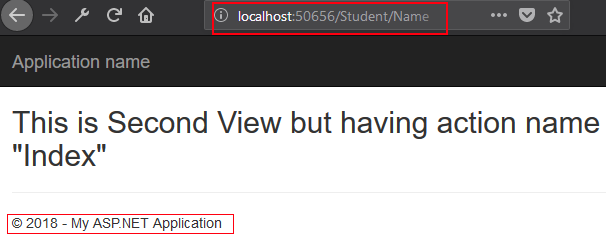
1. **public** **class** StudentController : Controller
2. {
3. **public** PartialViewResult Name()
4. {
5. **return** View("\_SecondView");
6. }
7. }

There is no need to make the View for “Name”. Because here the Name action method is calling a different View page. See the folder structure below,



The proper way of creating partial view page is, it should be used inside the regular view, having the name start with \_, and should reside in the Shared folder.

Now, when you execute it, you will see the same output but having content of layout page. See the figure below.



So here you can use PartialViewResult datatype.

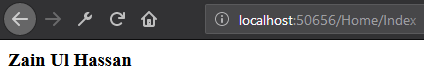
### **ContentResult**

"Content" result is a datatype which is responsible for the returning of content. But the point of interest is, you will have proper control over the returned content. I mean to say that you have to add the content in the “Content” helping method of ContentResult type. Here you can pass the contents like HTML content, Javascript content or any other content. Let’s understand it with examples.

Code of Home controller is given below,

1. **public** **class** HomeController : Controller
2. {
3. **public** ContentResult Index()
4. {
5. **return** Content("<h3>Zain Ul Hassan </h3>");
6. }
7. }

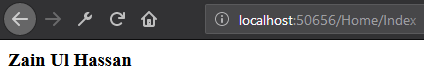
Its output will be,



You can also give MIME (Multipurpose Internet Mail Extensions) type in Content helper method to tell the MVC to take appropriate action after recognizing the content. Code is as shown below,

1. **public** **class** HomeController : Controller
2. {
3. **public** ContentResult Index()
4. {
5. **return** Content("<h3>Zain Ul Hassan</h3>", "text/html");
6. }
7. }

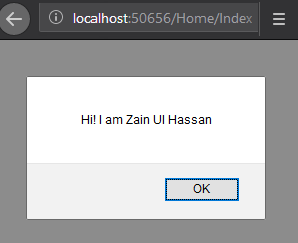
Output is the same as above,



You can also give Javascript content which will return by ContentResult type. Let’s take an example.

1. **public** **class** HomeController : Controller
2. {
3. **public** ContentResult Index()
4. {
5. **return** Content(
6. "<script> alert('Hi! I am Zain Ul Hassan') </script>"
7. );
8. }
9. }

The output will have an alert as shown below,



So you can use ContentResult datatype in the above way.

### **EmptyResult**

This type is genuinely for returning nothing. But the problem is, this EmptyResult type doesn’t have any helper method. So we can use it by making its object, as shown below,

1. **public** **class** HomeController : Controller
2. {
3. **public** EmptyResult Index()
4. {
5. **return** **new** EmptyResult();
6. }
7. }

But you want to use ActionResult then MVC will automatically perceive that it is of EmptyResult. For code, see below,

1. **public** **class** HomeController : Controller
2. {
3. **public** ActionResult Index()
4. {
5. **return** **null**;
6. }
7. }

So it is the use of EmptyResult type.

### **FileResult**

FileResult is a type which is used to return the file to the browser. The helper method for this type is File, and has many overloads. So if we use the following overload and specify only the URL of the file and the MIME (Multipurpose Internet Mail Extensions) type then we should see the content of the file in the browser. Let’s take an example.

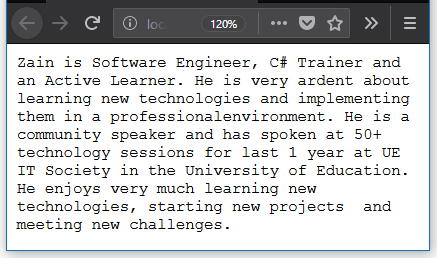
The overload we use is,

1. **protected** **internal** FilePathResult File(**string** fileName, **string** contentType);

And the code for action method using FileResult type is as follows,

1. **public** **class** HomeController : Controller
2. {
3. **public** FileResult Index()
4. {
5. **return** File("~/Files/text.txt", "text/plain");
6. }
7. }

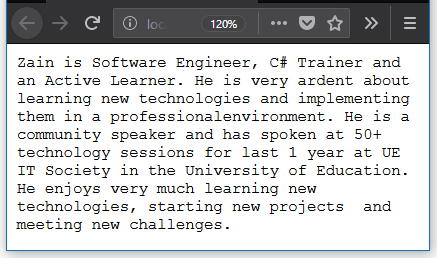
This file is included in the project and the URL is given above. And the output is given below,



Now if you want to return the file in byte array then you have to use following code,

1. **public** **class** HomeController : Controller
2. {
3. **public** FileResult Index()
4. {
5. **byte**[] fileBytes = System.IO.File.ReadAllBytes(Server.MapPath("~/Files/text.txt"));
6. **return** File(fileBytes, "text/plain");
7. }
8. }

The output is the same as above,



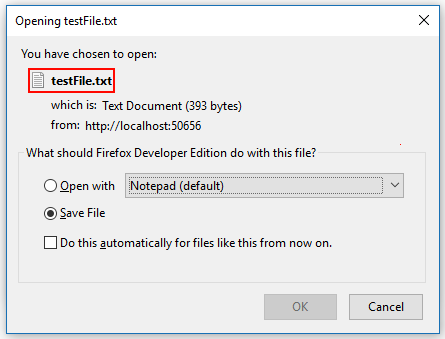
Now if you don’t want to show the file in the browser and want that your file should be asked to download then you have to use following overload the file helper method,

1. **protected** **internal** **virtual** FilePathResult File(**string** fileName, **string** contentType, **string** fileDownloadName);

The code of action using FileResult is given below,

1. **public** **class** HomeController : Controller
2. {
3. **public** FileResult Index()
4. {
5. **return** File(Url.Content("~/Files/text.txt"), "text/plain", "testFile.txt");
6. }
7. }

Then the output will be as shown below,



Hence it is the use of FileResult in different scenarios.

### **JsonResult**

It is the derived type of ActionResult which is used to represent the JSON data. To see how can we JSONify any data, you can see code below:

1. **public** JsonResult Index()
2. {
3. **return** Json(**new** { Name = "Zain Ul Hassan", ID = 1 });
4. }

Its output is as follows,



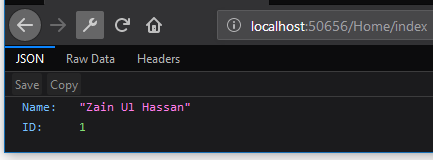
You will get this error after executing the above code. Now understand what the problem here is.

As you know, JSON contains your encoded data that can be precious, so MVC strictly stops the sharing of information over a GET request. In fact, MVC is trying to prevent you from JSON Hijacking.

So if you want to share the information over the GET request then you have to use the code shown below,

1. **public** JsonResult Index()
2. {
3. **return** Json(**new** { Name = "Zain Ul Hassan", ID = 1 }, JsonRequestBehavior.AllowGet);
4. }

The output will become,



So, JsonRequestBehavior.AllowGet permits the MVC to share the information over the GET request. But you should use the above feature when you are sure that your JSON data doesn’t contain sensitive data.

### **JavaScriptResult**

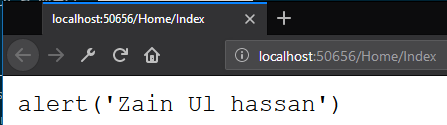
This derive type is used for returning the javascript code from the controller. When it executes we see the javascript code as we mentioned in the controller’s action method. For details, let’s take an example,

1. **public** JavaScriptResult Index()
2. {
3. **return** JavaScript("alert('Zain Ul hassan')");
4. }

And the View named as Index has following code,

1. <script type="text/javascript" src="@Url.Action("Index")"></script>

The output is as shown below,



## Redirection Results

This type of ActionResult is used for redirection purpose, which will you see with examples here. There are 2 types of redirection results,

* RedirectResult
* Redirect to Action Result

### **RedirectionResult**

If you use this type then it’ll redirect to the URL specified by you. An example is given below,

1. **public** RedirectResult Index()
2. {
3. **return** Redirect("https://www.c-sharpcorner.com/members/zain-ul-hassan2");
4. }

In the output, the specified URL will open, as shown below,



RedirectionResult is better for redirection to live URLs which means it is not a best practice to use it for redirection to the local pages of the current app. For this purpose use RedirectionToRouteResult as explained below,

### **RedirectionToRouteResult**

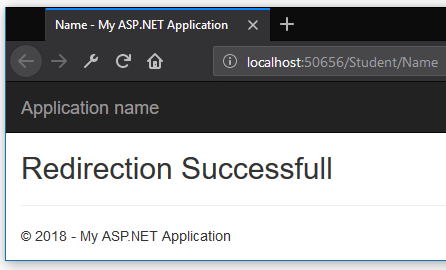
It is responsible for the redirection to the actions within the application. There are many helper methods in it which are actually overloads. We use RedirectToRoute which redirect us to the action within the specified controller. Let’s look at its example.

1. **public** RedirectToRouteResult Index()
2. {
3. **return** RedirectToRoute(**new** { controller = "Student", action = "Name" });
4. }

We have Name action method within the Student controller, and the code of Name view is given below,

1. <h2>Redirection Successfull</h2>

So the output is as follows,



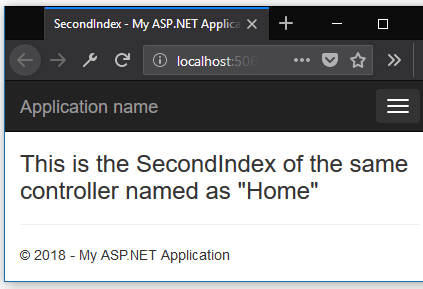
Now if you are in the same controller and don’t want to give the name of controller then the better option of is, RedirectToAction helper method. Look at the code below,

1. **public** **class** HomeController : Controller
2. {
3. **public** RedirectToRouteResult Index()
4. {
5. **return** RedirectToAction("SecondIndex");
6. }
7. **public** ActionResult SecondIndex()
8. {
9. **return** View();
10. }
11. }

And the View of SecondIndex’s code is below,

1. <h3>This is the SecondIndex of the same controller named as "Home"</h3>

The output is as follows,



As you can see, the helper method RedirectToAction is used for calling action within the current controller so if MVC doesn’t find that, 404 error will occur. RedirectToAction has many overloads, you can give controllers, actions, route values and much more. So you have to use according to your scenario.

## Status Result

Its responsibility is to give the status code to the browser. There are three types of this set explained below,

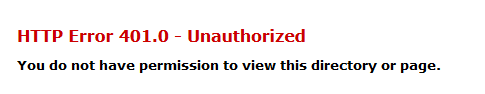
* HttpStatusCodeResult
* HttpUnauthorizedResult
* HttpNotFoundResult

### **HttpStatusCodeResult**

This type is used to give HTTP status code to the browser. Look at its example.

1. **public** HttpStatusCodeResult Index()
2. {
3. **return** **new** HttpStatusCodeResult(HttpStatusCode.Unauthorized);
4. }

The output is as shown below,



You can show your own message after using the overloaded version of the type HttpStatusCodeResult.

1. **public** HttpStatusCodeResult Index()
2. {
3. **return** **new** HttpStatusCodeResult(HttpStatusCode.Unauthorized, "Sorry! You don't have access.");
4. }

**Output**



Here HttpStatusCode is coming from System.Net namespace. And it contains all the HTTP status codes.

If you don’t want to use System.Net namespace then HttpStatusCodeResult has an overload whose output will be same as above. See the code below:

1. **public** HttpStatusCodeResult Index()
2. {
3. **return** **new** HttpUnauthorizedResult("Sorry! You don't have access.");
4. }

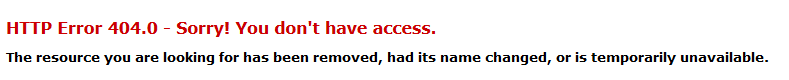
The output is the same as above but the code is looking more readable.

### **HttpNotFoundResult**

This is also an overload of HttpStatusCodeResult but it has the helper method, we don’t have to make an anonymous object. The code is as follows:

1. **public** HttpNotFoundResult Index()
2. {
3. **return** HttpNotFound("Sorry! You don't have access.");
4. }

**Output**



ASP.NET MVC - Actions

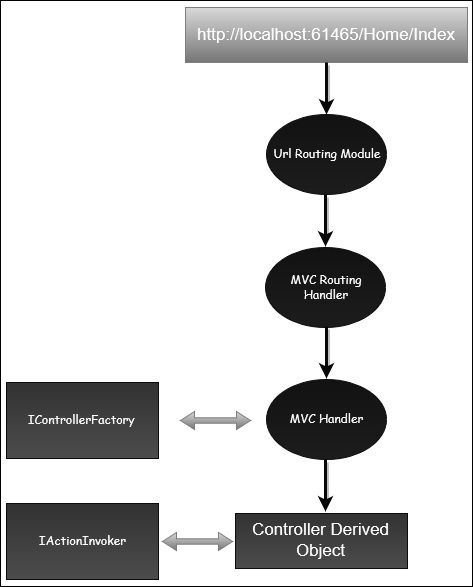
ASP.NET MVC Action Methods are responsible to execute requests and generate responses to it. By default, it generates a response in the form of ActionResult. Actions typically have a one-to-one mapping with user interactions.

For example, enter a URL into the browser, click on any particular link, and submit a form, etc. Each of these user interactions causes a request to be sent to the server. In each case, the URL of the request includes information that the MVC framework uses to invoke an action method. The one restriction on action method is that they have to be instance method, so they cannot be static methods. Also there is no return value restrictions. So you can return the string, integer, etc.

Request Processing

Actions are the ultimate request destination in an MVC application and it uses the controller base class. Let's take a look at the request processing.

* When a URL arrives, like /Home/index, it is the UrlRoutingModule that inspects and understands that something configured within the routing table knows how to handle that URL.



* The UrlRoutingModule puts together the information we've configured in the routing table and hands over control to the MVC route handler.
* The MVC route handler passes the controller over to the MvcHandler which is an HTTP handler.
* MvcHandler uses a controller factory to instantiate the controller and it knows what controller to instantiate because it looks in the RouteData for that controller value.
* Once the MvcHandler has a controller, the only thing that MvcHandler knows about is IController Interface, so it simply tells the controller to execute.
* When it tells the controller to execute, that's been derived from the MVC's controller base class. The Execute method creates an action invoker and tells that action invoker to go and find a method to invoke, find an action to invoke.
* The action invoker, again, looks in the RouteData and finds that action parameter that's been passed along from the routing engine.

Types of Action

Actions basically return different types of action results. The ActionResult class is the base for all action results. Following is the list of different kind of action results and its behavior.

|  |  |
| --- | --- |
| **Sr.No.** | **Name and Behavior** |
| 1 | **ContentResult**  Returns a string |
| 2 | **FileContentResult**  Returns file content |
| 3 | **FilePathResult**  Returns file content |
| 4 | **FileStreamResult**  Returns file content |
| 5 | **EmptyResult**  Returns nothing |
| 6 | **JavaScriptResult**  Returns script for execution |
| 7 | **JsonResult**  Returns JSON formatted data |
| 8 | **RedirectToResult**  Redirects to the specified URL |
| 9 | **HttpUnauthorizedResult**  Returns 403 HTTP Status code |
| 10 | **RedirectToRouteResult**  Redirects to different action/different controller action |
| 11 | **ViewResult**  Received as a response for view engine |
| 12 | **PartialViewResult**  Received as a response for view engine |

Let’s have a look at a simple example from the previous chapter in which we have created an EmployeeController.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace MVCControllerDemo.Controllers {

public class EmployeeController : Controller{

// GET: Employee

public ActionResult Search(string name){

var input = Server.HtmlEncode(name);

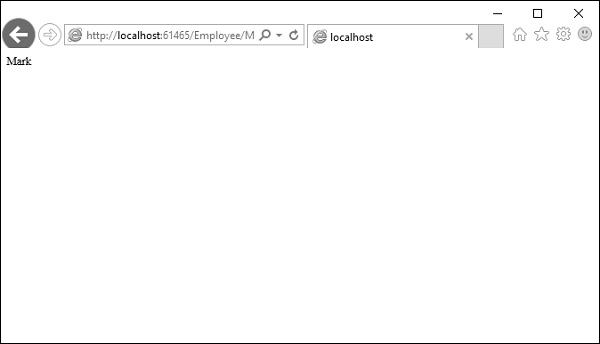
return Content(input);

}

}

}

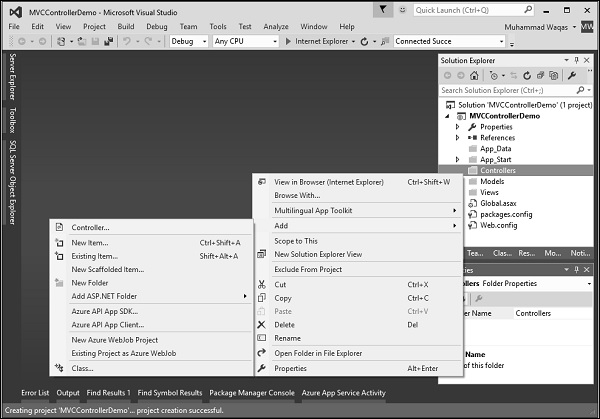
When you request the following URL **http://localhost:61465/Employee/Mark**, then you will receive the following output as an action.



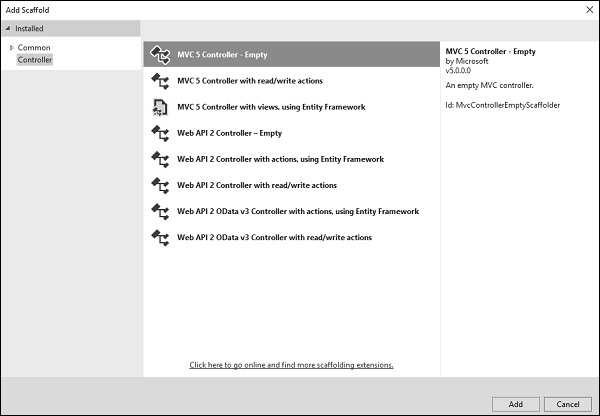
Add Controller

Let us add one another controller.

**Step 1** − Right-click on Controllers folder and select Add → Controller.



It will display the Add Scaffold dialog.



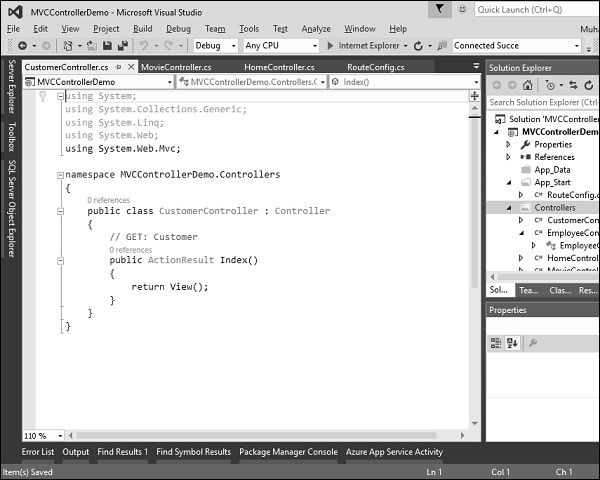
**Step 2** − Select the MVC 5 Controller – Empty option and click ‘Add’ button.

The Add Controller dialog will appear.



**Step 3** − Set the name to CustomerController and click ‘Add’ button.

Now you will see a new C# file ‘CustomerController.cs’ in the Controllers folder, which is open for editing in Visual Studio as well.



Similarly, add one more controller with name HomeController. Following is the HomeController.cs class implementation.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace MVCControllerDemo.Controllers {

public class HomeController : Controller{

// GET: Home

public string Index(){

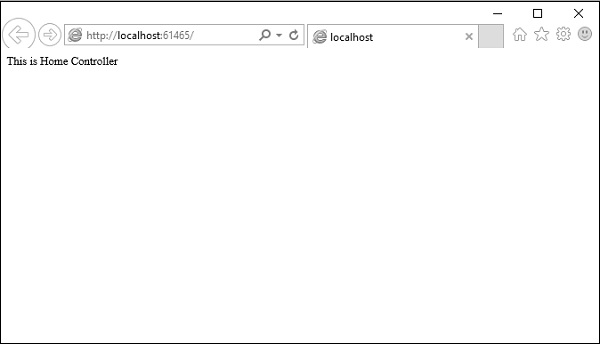
return "This is Home Controller";

}

}

}

**Step 4** − Run this application and you will receive the following output.



**Step 5** − Add the following code in Customer controller, which we have created above.

public string GetAllCustomers(){

return @"<ul>

<li>Ali Raza</li>

<li>Mark Upston</li>

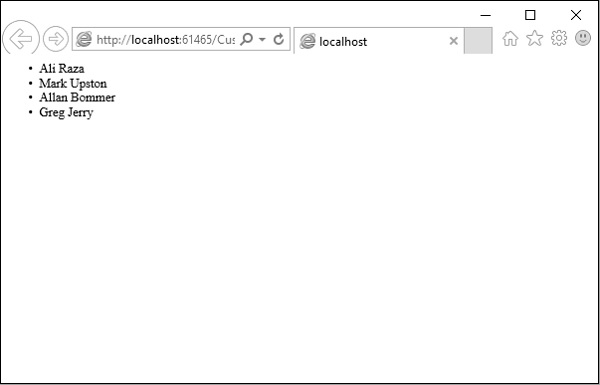
<li>Allan Bommer</li>

<li>Greg Jerry</li>

</ul>";

}

**Step 6** − Run this application and request for **http://localhost:61465/Customer/GetAllCustomers**. You will see the following output.



You can also redirect to actions for the same controller or even for a different controller.

Following is a simple example in which we will redirect from HomeController to Customer Controller by changing the code in HomeController using the following code.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace MVCControllerDemo.Controllers{

public class HomeController : Controller{

// GET: Home

public ActionResult Index(){

return RedirectToAction("GetAllCustomers","Customer");

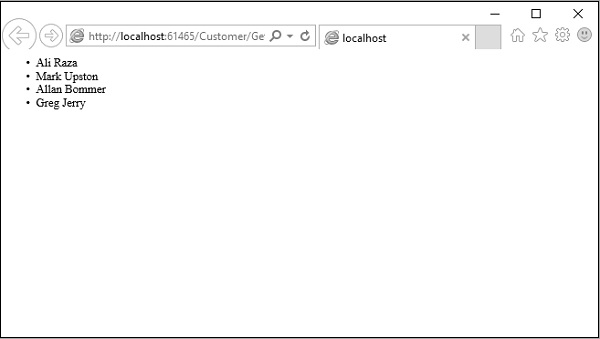
}

}

}

As you can see, we have used the RedirectToAction() method ActionResult, which takes two parameters, action name and controller name.

When you run this application, you will see the default route will redirect it to /Customer/GetAllCustomers



[**Previous Page**](https://www.tutorialspoint.com/asp.net_mvc/asp.net_mvc_controllers.htm)[**Print Page**](https://www.tutorialspoint.com/asp.net_mvc/asp.net_mvc_actions.htm)[**Next Page**](https://www.tutorialspoint.com/asp.net_mvc/asp.net_mvc_filters.htm)

Advertisements