

MVC :: Understanding Controllers, Controller Actions, and Action Results

This tutorial explores the topic of ASP.NET MVC controllers, controller actions, and action results. After you complete this tutorial, you will understand how controllers are used to control the way a visitor interacts with an ASP.NET MVC website.

Understanding Controllers

MVC controllers are responsible for responding to requests made against an ASP.NET MVC website. Each browser request is mapped to a particular controller. For example, imagine that you enter the following URL into the address bar of your browser:

<http://localhost/Product/Index/3>

In this case, a controller named `ProductController` is invoked. The `ProductController` is responsible for generating the response to the browser request. For example, the controller might return a particular view back to the browser or the controller might redirect the user to another controller.

You create a new controller by adding a new controller to your ASP.NET MVC application's Controllers folder. Right-click the Controller's folder and select the menu option `Add, New Item` and select the MVC Controller Class template (see Figure 1). A controller name must include the suffix *Controller*. For example, the controller name `ProductController` is fine but the controller name `Product` won't work.

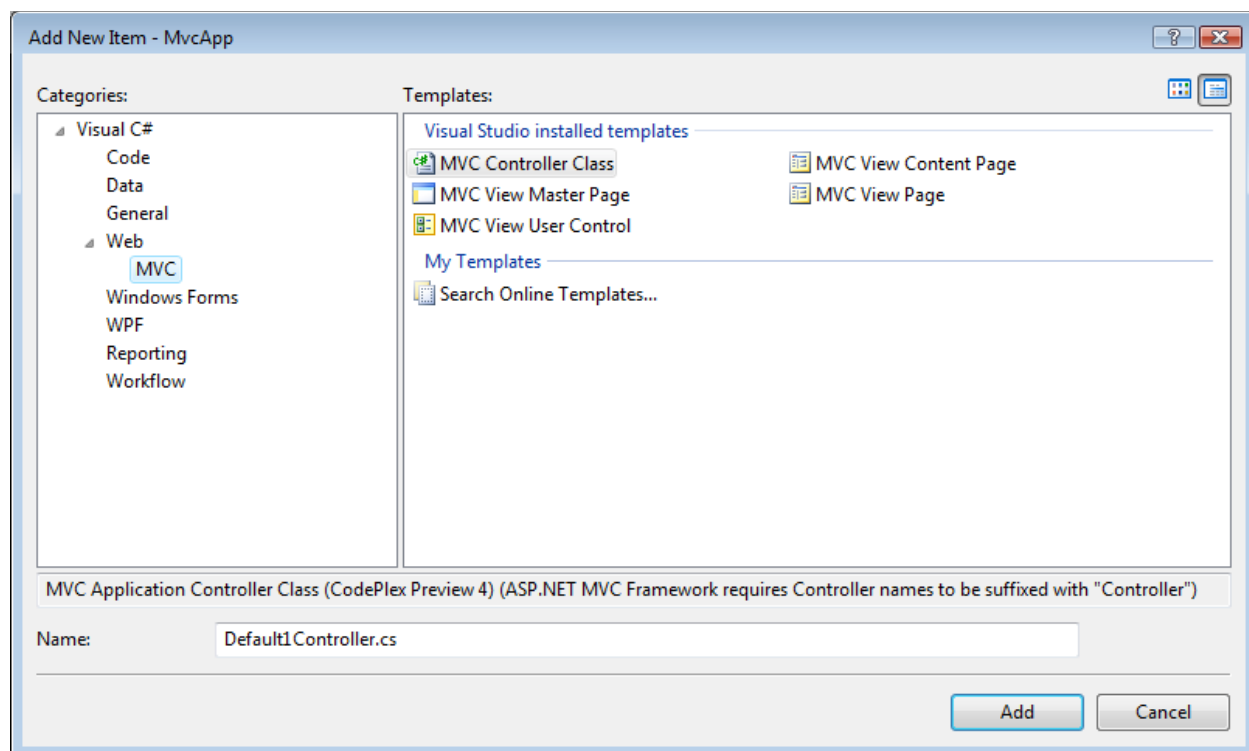


Figure 1 – Creating a new Controller

If you create a new controller named `ProductController` then you will get the file in Listing 1.

Listing1 – `ProductController.cs`

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MvcApp.Controllers
{
    public class ProductController : Controller
    {
        public ActionResult Index()
        {
            // Add action logic here
            throw new NotImplementedException();
        }
    }
}
```

As you can see from Listing 1, a controller is just a class (a Visual Basic .NET or C# class). A controller is a class that derives from the base `System.Web.Mvc.Controller` class. Because a controller inherits from this base class, a controller inherits several useful methods for free (We discuss these methods in a moment).

Understanding Controller Actions

A controller exposes controller actions. An action is a method on a controller that gets called when you enter a particular URL in your browser address bar. For example, imagine that you make a request for the following URL:

<http://localhost/Product/Index/3>

In this case, the `Index()` method is called on the `ProductController` class. The `Index()` method is an example of a controller action.

A controller action must be a public method of a controller class. C# methods, by default, are private methods. Realize that any public method that you add to a controller class is

exposed as a controller action automatically (You must be careful about this since a controller action can be invoked by anyone in the universe simply by typing the right URL into a browser address bar).

There are some additional requirements that must be satisfied by a controller action. A method used as a controller action cannot be overloaded. Furthermore, a controller action cannot be a static method. Other than that, you can use just about any method as a controller action.

Understanding Action Results

A controller action returns something called an *action result*. An action result is what a controller action returns in response to a browser request.

The ASP.NET MVC framework supports six standard types of action results:

1. `ViewResult` – Represents HTML and markup.
2. `EmptyResult` – Represents no result.
3. `RedirectResult` – Represents a redirection to a new URL.
4. `RedirectToRouteResult` – Represents a redirection to a new controller action.
5. `JsonResult` – Represents a JavaScript Object Notation result that can be used in an AJAX application.
6. `ContentResult` – Represents a text result.

All of these action results inherit from the base `ActionResult` class.

In most cases, a controller action returns a `ViewResult`. For example, the Index controller action in Listing 2 returns a `ViewResult`.

Listing 2 – `BookController.cs`

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MvcApp.Controllers
{
    public class BookController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

```
}  
  
}
```

When an action returns a `ViewResult`, HTML is returned to the browser. The `Index()` method in Listing 2 returns a view named `Index.aspx` to the browser.

Notice that the `Index()` action in Listing 2 does not return a `ViewResult()`. Instead, the `View()` method of the Controller base class is called. Normally, you do not return an action result directly. Instead, you call one of the following methods of the Controller base class:

1. `View` – Returns a `ViewResult` action result.
2. `Redirect` – Returns a `RedirectResult` action result.
3. `RedirectToAction` – Returns a `RedirectToRouteResult` action result.
4. `RedirectToRoute` – Returns a `RedirectToRouteResult` action result.
5. `Json` – Returns a `JsonResult` action result.
6. `Content` – Returns a `ContentResult` action result.

So, if you want to return a View to the browser, you call the `View()` method. If you want to redirect the user from one controller action to another, you call the `RedirectToAction()` method. For example, the `Details()` action in Listing 3 either displays a view or redirects the user to the `Index()` action depending on whether the `Id` parameter has a value.

Listing 3 – CustomerController.cs

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Web;  
using System.Web.Mvc;  
  
namespace MvcApp.Controllers  
{  
    public class CustomerController : Controller  
    {  
  
        public ActionResult Details(int? Id)  
        {  
            if (Id == null)  
                return RedirectToAction("Index");  
            return View();  
        }  
    }  
}
```

```

        public ActionResult Index()
        {
            return View();
        }
    }
}

```

The `ContentResult` action result is special. You can use the `ContentResult` action result to return an action result as plain text. For example, the `Index()` method in Listing 4 returns a message as plain text and not as HTML.

Listing 4 – `StatusController.cs`

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MvcApp.Controllers
{
    public class StatusController : Controller
    {
        public ContentResult Index()
        {
            return Content("Hello World!");
        }
    }
}

```

When the `StatusController.Index()` action is invoked, a view is not returned. Instead, the raw text "Hello World!" is returned to the browser.

If a controller action returns a result that is not an action result – for example, a date or an integer – then the result is wrapped in a `ContentResult` automatically. For example, when the `Index()` action of the `WorkController` in Listing 5 is invoked, the date is returned as a `ContentResult` automatically.

Listing 5 – WorkController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MvcApp.Controllers
{
    public class WorkController : Controller
    {
        public DateTime Index()
        {
            return DateTime.Now;
        }
    }
}
```

The `Index()` action in Listing 5 returns a `DateTime` object. The ASP.NET MVC framework converts the `DateTime` object to a string and wraps the `DateTime` value in a `ContentResult` automatically. The browser receives the date and time as plain text.

Summary

The purpose of this tutorial was to introduce you to the concepts of ASP.NET MVC controllers, controller actions, and controller action results. In the first section, you learned how to add new controllers to an ASP.NET MVC project. Next, you learned how public methods of a controller are exposed to the universe as controller actions. Finally, we discussed the different types of action results that can be returned from a controller action. In particular, we discussed how to return a `ViewResult`, `RedirectToActionResult`, and `ContentResult` from a controller action.