Image by Ron Reiring

As noted in Routing Basics in ASP.NET MVC, routing in ASP.NET Web API is functionally very similar to the standard MVC routing. Understanding the MVC routing model will provide most of what you need to know to leverage that knowledge against the Web API framework, so long as you keep in mind the key differences between the two.

The differences between the two are largely dictated by the different consumption models driving each type of application.

IMPORANT NOTE: *This post covers the most basic and fundamental concepts of routing as applied to the ASP.NET Web API framework. The target audience are those with little or no familiarity with routing in general, or who may be looking to understand the most basic differences between routing in ASP.NET MVC and Web API.*

If you have little to no knowledge of MVC routing, you may want to review my post on MVC routing basics first.

In my next post, having covered the fundamentals, I will examine route customization.

Web API and MVC – Two Separate Frameworks

Because the Web API grew (in part) out of the efforts of

the ASP.NET MVC team, and because the default Web API project template in Visual Studio is bundled up inside an MVC application, it is easy to think that the Web API framework is some kind of subset of the MVC framework. In fact, the two inter-operate together very well, but that is by design. They are, in fact, separate frameworks which adhere to some common architectural patterns.

Instead of paraphrasing the overview of Web API and the architectural goals of the project, I will let the MVC team do so, because honestly, they know better than I do! from the most helpful book Professional ASP.NET MVC 4 authored by (mostly) the MVC team:

> *"ASP.NET MVC excels at accepting form data and generating HTML; ASP.NET Web API excels at accepting and generating structured data like JSON and XML."*
> *MVC has flirted with providing structured data support (with JsonResult and the JSON value provider), but it still fell short in several ways that are important to API programmers, including:*
>
> - Dispatching to actions based on HTTP verbs rather than action names
> - Accepting and generating content which may not necessarily be object

> oriented (not only XML, but also content like images, PDF files, or VCARDs
>
> - Content type negotiation, which allows the developer to both accept and genereate structured content independent of its wire representation
> - Hosting outside of the ASP.NET runtime stack and IIS web server, something which WCF has been unable to do for years
>
> *The Web API team went to great lengths to try to allow you to leverage your existing ASP.NET experience with controllers, actions, filters, model binders, dependency injection, and the like. Many of these same concepts appear in Web API in very similar forms, which make applications that combine MVC and Web API seem very well integrated."*

As noted above, one of the key differences between MVC and Web API is that Web API returns an appropriate representation of the model object as a result. A related difference is that Web API performs content type negotiation in response to incoming requests, and attempts to return the proper content type (JSON in response to an incoming request specifying JSON as the preferred content resource representation,

for example).

Suffice it to say that Web API is an effort to create a mechanism for the efficient exchange of data and other content using HTTP. This effort is reflected in the architecture of the framework, and results in some of the differences in routing between the two frameworks.

Web API Route Configuration Files

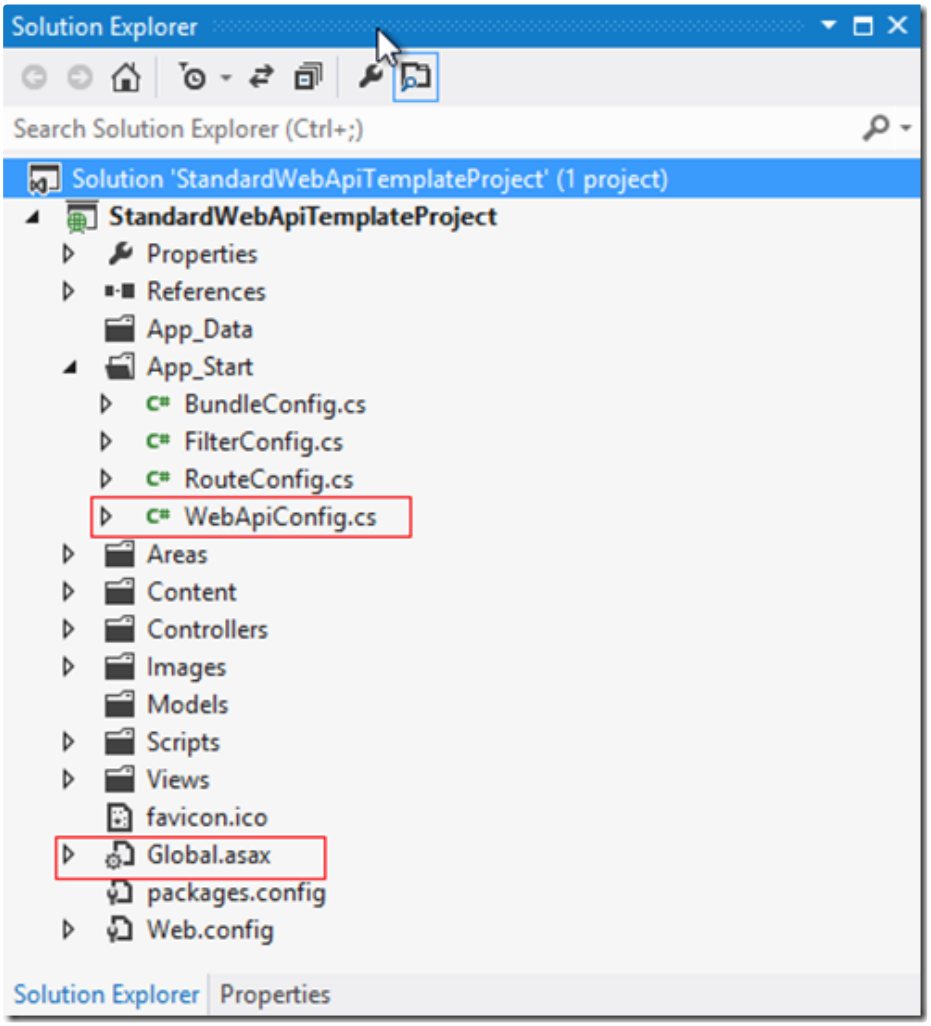
The default Visual Studio Web API project template is rather large, as it incorporates a complete MVC project as well. The resulting project has some pretty cool features, including automatic documentation for the API you build (that is the purpose of the MVC site contained in the project), but also presents a rather bloated project file, with a large number of folders and dependencies.

NOTE: To see how to create a stripped-down Web API project template in Visual Studio, see Creating a Clean, Minimal-Footprint ASP.NET WebAPI Project with VS 2012 and ASP.NET MVC 4

As with a standard MVC project, the route configuration is called from the `Global.asx` file. However, in this standard Web API project, there exists both a `RouteConfig` class, and a `WebApiConfig` class, both in the *App_Start* folder:

Global.asx File and the WebApiConfig File in a

Typical MVC Project:



The `RouteConfig.cs` file here is the same as in any MVC project, and sets up routes for the MVC framework. The `WebApiConfig.cs` file is where our Web API routing configuration happens. However, if we open the file, it looks pretty familiar:

The WebApiConfig Class and MapHttpRoute Method:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Http;
namespace StandardWebApiTemplateProject
{
```

```
    public static class WebApiConfig
    {
        public static void Register(HttpConfiguratic
        {
            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{ic
                defaults: new { id = RouteParameter.
            );
        }
    }
}
```

There are a few important differences here, though. First, notice that in the using statements at the top of the file, there are no references to `System.Web.Mvc` but instead, we find a reference to `System.Web.Http`. That's because the Web API team re-created the routing functionality in a library with no dependencies on ASP.NET or MVC. Also notice that instead of calling `Routes.MapRoutes` as in the MVC `RouteConfig` class, we instead call **`Config.Routes.MapHttpRoutes`**.

And, as we can see, the default route template for Web API looks a little different.

How Routing Works for Web API

Unlike ASP.NET MVC, the Web API routing convention routes incoming requests to a specific controller, but by default simply matches the HTTP verb of the request to an action method whose name begins with that verb. Recall the default MVC route configuration:

The Default MVC Route Template:

```
{controller}/{action}/{id}
```

In contrast, the default Web API route template looks like this:

The Default Web API Route Template:

```
api/{controller}/{id}
```

The literal `api` at the beginning of the Web API route template above makes it distinct from the standard MVC route. Also, it is a good convention to include as part of an API route a segment that lets the consumer know they are accessing an API instead of a standard site.

Also notice that unlike the familiar MVC route template, the Web API template does not specify an `{action}` route parameter. This is because, as we mentioned earlier, the Web API framework will by default map incoming requests to the appropriate action based upon the HTTP verb of the request.

Consider the following typical Web API Controller:

WebApi Controller from Default Visual Studio WebApi Template

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Web.Http;
```

```
namespace StandardWebApiTemplateProject.Controllers
{
    public class ValuesController : ApiController
    {
        // GET api/values
        public IEnumerable<string> Get()
        {
            return new string[] { "value1", "value2"
        }
        // GET api/values/5
        public string Get(int id)
        {
            return "value";
        }
        // POST api/values
        public void Post([FromBody]string value)
        {
        }
        // PUT api/values/5
        public void Put(int id, [FromBody]string val
        {
        }
        // DELETE api/values/5
        public void Delete(int id)
        {
        }
    }
}
```

First, take note that Web API controllers do not inherit from `System.Web.Mvc.Controller`, but instead from `System.Web.Http.Controller`. Again, we are working with a familiar, but distinct library.

Of particular note, however, are the default method names (the example above is what is created as part of the default VS 2012 project template). Again, without specifying a specific method using an `{action}` route parameter, the Web API framework determines the appropriate route based on the HTTP verb of the

incoming request, and calls the proper method accordingly.

The following incoming URL, included as part of an HTTP GET message, would map to the first `Get()` method as defined in the controller above:

Example URL With No ID Parameter

```
http://mydomain/values/
```

Similarly, this URL, also as part of an HTTP GET request, would map to the second `Get(id)` method:

Example URL Including ID Parameter

```
http://mydomain/values/5
```

We could, however, modify our controller thusly, and the routing would still work without modification:

Modifications to the ValuesController Class:

```
// GET api/values
public IEnumerable<string> GetValues()
{
    return new string[] { "value1", "value2" };
}
// GET api/values/5
public string GetValue(int id)
{
    return "value";
}
// POST api/values
public void PostValue([FromBody]Book value)
{
}
// PUT api/values/5
public void PutValue(int id, [FromBody]string value)
```

```
{
}
// DELETE api/values/5
public void DeleteValue(int id)
{
}
```


to all of our method names. Under these circumstances, the unmodified route parameter will still work just fine. When per convention, no `{action}` route parameter is specified, the Web API framework again appends the "Controller" suffix to the value provided for the `{controller}` parameter, and then scans the project for a suitably named class (in this case, one which derives from `ApiController`).

Action Method Selection

Once the proper controller class is selected, the framework examines the HTTP action verb of the request, and searches the class for method names with a matching prefix.

In order to determine the proper method, the framework then examines the additional URL parameters and attempts to match them with method arguments by name (case insensitive). The method with the most matching arguments will be selected.

An item to note here. Unlike in MVC, in Web API complex types are not allowed as part of the URL. Complex types

must be placed in the body of the HTTP message. Also, there can be one, and only one such complex type in the message body.

In evaluating parameter matches against method arguments, any complex types and URL query strings are disregarded when searching for a match.

Web API and Action Naming

We can modify our default Web API route to include an `{action}` route parameter, in which case selection will occur similar to that in MVC. However, action methods defined on our controller still need to include the proper HTTP action verb prefix, and incoming URLs must use the full method name.

If we modify our default route thusly, adding an `{action}` parameter:

The Modified WebApiConfig Class and MapHttpRoute Method:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Http;
namespace StandardWebApiTemplateProject
{
    public static class WebApiConfig
    {
        public static void Register(HttpConfiguratic
        {
            config.Routes.MapHttpRoute(
                name: "DefaultApi",
```

```
                routeTemplate: "api/{controller}/{ac
                defaults: new { id = RouteParameter.
            );
        }
    }
}
```

To see how this works, consider the incoming URL:

```
http://mydomain/values/5
```

This will no longer work. If we wish to make a request of our modified `ValuesController` now, we would need to submit:

```
http://mydomain/values/GetValue/5
```

Review of the Basics for Web API Routing

- The routing convention for Web API is to route URLs to a controller, and then to the action which matches the HTTP verb of the request message. Action methods on the controller must either match the HTTP action verb, or at least include the action verb as a prefix for the method name.
- The default route template for a Web API Project is {controller}/{id} where the {id} parameter is optional.
- Web API route templates may optionally

include an {action} parameter. However, the action methods defined on the controller must be named with the proper HTTP action verb as a prefix in order for the routing to work.

- In matching incoming HTTP messages to controllers, the Web API framework identifies the proper controller by appending the literal "Controller" to the value of the {controller} route parameter, then scans the project for a class matching that name.
- Actions are selected from the controller by considering the non-complex route parameters, and matching them by name to the arguments of each method which matches the HTTP verb of the request. The method which matches the most parameters is selected.
- Unlike MVC, URLs in Web API cannot contain complex types. Complex types must be placed in the HTTP message body. There may be one, and only one complex type in the body of an HTTP message.

That was Pretty Darn Basic. What Next?

The above represents a very simple overview of the Web API routing basics, focusing on the default configurations and conventions. In the next post we will examine more

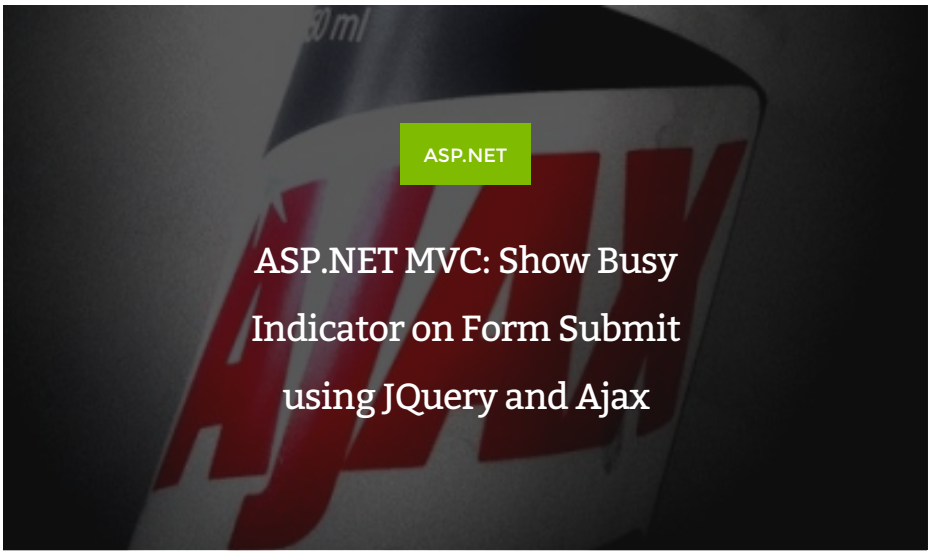
advanced routing considerations, and route customization.

Additional Resources

- Routing Basics in ASP.NET MVC
- Route Customization in ASP.NET MVC
- Creating a Clean, Minimal-Footprint ASP.NET WebAPI Project with VS 2012 and ASP.NET MVC 4
- Building Out a Clean, REST-ful Web Api Service with a Minimal Web Api Project
- Routing in ASP.NET Web API

ASP.NET C# ROUTING WEB API

RELATED ARTICLES



VIEW COMMENTS

PREVIOUS POST
Routing Basics in ASP.NET MVC

NEXT POST

C# – Wildcard Search Using LINQ