**Routing Tables**

To determine which action to invoke, the framework uses a ***routing table***. The Visual Studio project template for Web API creates a default route:

routes.MapHttpRoute(

name: "API Default",

routeTemplate: "api/{controller}/{id}",

defaults: new { id = RouteParameter.Optional }

);

This route is defined in the WebApiConfig.cs file, which is placed in the App\_Start directory.

For more information about the **WebApiConfig** class, see [Configuring ASP.NET Web API](http://www.asp.net/web-api/overview/extensibility/configuring-aspnet-web-api).

If you self-host Web API, you must set the routing table directly on the **HttpSelfHostConfiguration** object. For more information, see [Self-Host a Web API](http://www.asp.net/web-api/overview/hosting-aspnet-web-api/self-host-a-web-api).

Each entry in the routing table contains a *route template*. The default route template for Web API is "api/{controller}/{id}". In this template, "api" is a literal path segment, and {controller} and {id} are placeholder variables.

When the Web API framework receives an HTTP request, it tries to match the URI against one of the route templates in the routing table. If no route matches, the client receives a 404 error. For example, the following URIs match the default route:

* /api/contacts
* /api/contacts/1
* /api/products/gizmo1

However, the following URI does not match, because it lacks the "api" segment:

* /contacts/1

**Note:** The reason for using "api" in the route is to avoid collisions with ASP.NET MVC routing. That way, you can have "/contacts" go to an MVC controller, and "/api/contacts" go to a Web API controller. Of course, if you don't like this convention, you can change the default route table.

Once a matching route is found, Web API selects the controller and the action:

* To find the controller, Web API adds "Controller" to the value of the *{controller}* variable.
* To find the action, Web API looks at the HTTP method, and then looks for an action whose name begins with that HTTP method name. For example, with a GET request, Web API looks for an action that starts with "Get...", such as "GetContact" or "GetAllContacts".  This convention applies only to GET, POST, PUT, and DELETE methods. You can enable other HTTP methods by using attributes on your controller. We’ll see an example of that later.
* Other placeholder variables in the route template, such as *{id},* are mapped to action parameters.

Let's look at an example. Suppose that you define the following controller:

public class ProductsController : ApiController

{

public void GetAllProducts() { }

public IEnumerable<Product> GetProductById(int id) { }

public HttpResponseMessage DeleteProduct(int id){ }

}

Here are some possible HTTP requests, along with the action that gets invoked for each:

|  |  |  |  |
| --- | --- | --- | --- |
| HTTP Method | URI Path | Action | Parameter |
| GET | api/products | GetAllProducts | *(none)* |
| GET | api/products/4 | GetProductById | 4 |
| DELETE | api/products/4 | DeleteProduct | 4 |
| POST | api/products | *(no match)* |  |

Notice that the *{id}* segment of the URI, if present, is mapped to the *id* parameter of the action. In this example, the controller defines two GET methods, one with an *id* parameter and one with no parameters.

Also, note that the POST request will fail, because the controller does not define a "Post..." method.