Web API

By

Srikanth Pragada

What is Web API?

- HTTP is not just for serving up web pages. It is also a powerful platform for building APIs that expose services and data.
- HTTP is simple, flexible, and ubiquitous.
- HTTP services can reach a broad range of clients, including browsers, mobile devices, and traditional desktop applications.
- ASP.NET Web API is a framework for building web APIs on top of the .NET Framework.
- Web API is based on Routing, Controller and Model features.

HTTP methods in Web API

- Methods in Web API are called based on HTTP request method sent from client.
- The following are 4 different request methods and 5 URLS that are generally supported by Web API.
- However, some Web APIs only support a few HTTP methods. For example, read-only Web API only support GET method and no other request methods.

Http Method	Purpose	Method Signature	URL
GET	Get all objects	<datatype> Get()	/messages
GET	Get only one object by id	<datatype> Get(id)	/messages/10
POST	Insert a new object	void Post(object)	/messages
PUT	Updates an existing object	void Put(id, newobject)	/messages/10
DELETE	Deletes an existing object	void Delete(id)	/messages/10

Registering WEB API – global.asax

We need to add the following code for Application_Start event of Global.asax to register the routing information regarding Web API.

```
// Global.asax
public class WebApiApplication : System.Web.HttpApplication
{
        protected void Application_Start()
        {
            AreaRegistration.RegisterAllAreas();
            GlobalConfiguration.Configure(WebApiConfig.Register);
            FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
            RouteConfig.RegisterRoutes(RouteTable.Routes);
            BundleConfig.RegisterBundles(BundleTable.Bundles);
        }
}
```

Registering WEB API – WebApiConfig.cs

Provide routing information regarding Web API in WebApiConfig.cs, which is in **App_Start** folder.

```
public static class WebApiConfig
{
      public static void Register(HttpConfiguration config)
      {
            // Web API routes
            config.MapHttpAttributeRoutes();

            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );
      }
}
```

Adding Controller

- A controller is an object that handles HTTP requests.
- Control must inherit **ApiController** class and provide methods to handle different Http request methods.

```
usingSystem.Collections.Generic;
usingSystem.Net.Http;
usingSystem.Web.Http;

public class MessagesController : ApiController {
 public string[] messages = {
    "A place for everything, everything in its place",
    "Work is Worship",
    "Failing to plan is planning to fail" };

 public string Get(int id) {
    return messages[id];
 }
 public IList<string> Get(){
    return messages.ToList<string>();
 }
}
```

Adding Client – HTML page with jQuery

- A client for Web API can be anything that can make HTTP request.
- If a GET request is made then all messages are sent from server to client.
- If a GET request is made with message id as route parameter then only one message with that ID is passed from server to client.

```
<head>
<script type="text/javascript"  src="jquery.js"></script>
<script type="text/javascript">
function getMessage() {
      $.getJSON("api/messages/" + $("#msgid").val(), null,
                  function (message) {
                        alert(message);
                  }
            );
 }
</script> </head>
<body>
Message Id :<input id="msgid" type="text" />
<p />
<input type="button" value="Get Message" onclick="getMessage()" />
</body>
```

Attributes related to Http Methods

- Web API also selects actions based on the HTTP method of the request (GET, POST, etc).
- By default, Web API looks for a case-insensitive match with the start of the controller method name.
- For example, a controller method named PutCustomers matches an HTTP PUT request.
- You can override this convention by decorating the method with any the following attributes.

  - `[HttpDelete]`
  - `[HttpGet]`
  - `[HttpHead]`
  - `[HttpOptions]`
  - `[HttpPost]`
  - `[HttpPut]`

Attribute Based Routing

- Web API 1 used convention-based routing
- Web API 2 introduced attribute based routing
- In convention-based routing templates are defined in a single place and the routing rules are applied consistently across all controllers.
- Attribute based routing is enabled by using **MapHttpAttributeRoutes**() method
- Routes to be associated with methods are provide by **Route** attribute

```
public static class WebApiConfig {
        public static void Register(HttpConfiguration config) {
            // Web API routes
            config.MapHttpAttributeRoutes();

            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );
        }
}
```

Attribute Based Routing - MessagesController

```
public class MessagesController : ApiController
    {
        public string[] messages =
            {"A place for everything, everything in its place",
             "Work is Worship",
             "Failing to plan is planning to fail" };

        [Route("Messages")]
        [HttpGet]
        public IEnumerable<string> Get()
        {
            return messages;
        }
        [Route("Messages/{id}")]
        [HttpGet]
        public string Get(int id)
        {
            return messages[id];
        }
```

Attribute Based Routing – URLs and Results

http://localhost:58027/Messages

```
<ArrayOfstring xmlns:i="http://www.w3.org/2001/XMLSchema-
instance" xmlns="http://schemas.microsoft.com/2003/10/Serialization/Arrays">
<string>A place for everything, everything in its place</string>
<string>Work is Worship</string>
<string>Failing to plan is planning to fail</string>
</ArrayOfstring>
```

http://localhost:58027/Messages/1

```
<string xmlns="http://schemas.microsoft.com/2003/10/Serialization/">Work is
Worship</string>
```

Attribute Based Routing – More Examples

```
Route("customers/{customerId}/orders")]                    // customers/101/orders
[HttpGet]
public IEnumerable<Order> FindOrdersByCustomer(int customerId) { ... }
```

```
Route("customers/{customerId}/orders/{startDate}")]        // customers/101/orders/20190401
[HttpGet]
public IEnumerable<Order> FindOrdersByCustomer(int customerId, string startDate) { ... }
```

Content Negotiation

- Web API implements content negotiation
- The HTTP specification (RFC 2616) defines content negotiation as "the process of selecting the best representation for a given response when there are multiple representations available."
- The primary mechanism for content negotiation in HTTP are these request headers:
  - ✓ **Accept:** Which media types are acceptable for the response, such as "application/json," "application/xml," or a custom media type such as "application/vnd.example+xml"
  - ✓ **Accept-Charset:** Which character sets are acceptable, such as UTF-8 or ISO 8859-1.
  - ✓ **Accept-Encoding:** Which content encodings are acceptable, such as gzip.
  - ✓ **Accept-Language:** The preferred natural language, such as "en-us".
- The server can also look at other portions of the HTTP request. For example, if the request contains an X-Requested-With header, indicating an AJAX request, the server might default to JSON if there is no Accept header.

RoutePrefix Attribute

We can set a common prefix for an entire controller by using the [RoutePrefix] attribute.

```
[RoutePrefix("api/books")]
public class BooksController : ApiController
{
    // GET api/books
    [Route("")]
    public IEnumerable<Book> Get() { ... }

    // GET api/books/5
    [Route("{id:int}")]
    public Book Get(int id) { ... }

    // POST api/books
    [Route("")]
    public HttpResponseMessage Post(Book book) { ... }
}
```

Route Constraints

Route constraints let you restrict how the parameters in the route template are matched.

```
{parameter:constraint}
```

Constraint	Description	Example
alpha	Matches uppercase or lowercase Latin alphabet characters (a-z, A-Z)	{x:alpha}
bool	Matches a Boolean value.	{x:bool}
datetime	Matches a DateTime value.	{x:datetime}
decimal	Matches a decimal value.	{x:decimal}
double	Matches a 64-bit floating-point value.	{x:double}
float	Matches a 32-bit floating-point value.	{x:float}
guid	Matches a GUID value.	{x:guid}
int	Matches a 32-bit integer value.	{x:int}
length	Matches a string with the specified length or within a specified range of lengths.	{x:length(6)} {x:length(1,20)}
long	Matches a 64-bit integer value.	{x:long}
max	Matches an integer with a maximum value.	{x:max(10)}
maxlength	Matches a string with a maximum length.	{x:maxlength(10)}
min	Matches an integer with a minimum value.	{x:min(10)}
minlength	Matches a string with a minimum length.	{x:minlength(10)}
range	Matches an integer within a range of values.	{x:range(10,50)}
regex	Matches a regular expression.	{x:regex(^\d{3}-\d{3}-\d{4}$)}

Route Constraints - Examples

- Some of the constraints, such as "min", take arguments in parentheses. You can apply multiple constraints to a parameter, separated by a colon.
- You can make a URI parameter optional by adding a question mark to the route parameter.
- If a route parameter is optional, you must define a default value for the method parameter.

```
[Route("books/{id:int:min(1)}")]
public User GetUserById(int id) { ... }
```

```
[Route("books/subject/{sub:string?}")]
public IEnumerable<Book> GetBooks(string sub = "aspnet") { ... }
```

HttpResponseException class

- It is used to throw an exception from Web API with the specified status code.

```
HttpResponseException(HttpResponseMessage)
HttpResponseException(HttpStatusCode)
```

HttpStatusCode Enum

Contains the values of status codes defined for HTTP.

OK	200	Equivalent to HTTP status 200. OK indicates that the request succeeded and that the requested information is in the response. This is the most common status code to receive.
Created	201	Equivalent to HTTP status 201. Created indicates that the request resulted in a new resource created before the response was sent.
NoContent	204	Equivalent to HTTP status 204. NoContent indicates that the request has been successfully processed and that the response is intentionally blank.
BadRequest	400	Equivalent to HTTP status 400. BadRequest indicates that the request could not be understood by the server. BadRequest is sent when no other error is applicable, or if the exact error is unknown or does not have its own error code.
NotFound	404	Equivalent to HTTP status 404. NotFound indicates that the requested resource does not exist on the server.
InternalServerError	500	Equivalent to HTTP status 500. InternalServerError indicates that a generic error has occurred on the server.

HttpResponseMessage - Example

```
// Returns an object of Product, if found
// Throws HttpResponseException, if product is not found

public Product GetProduct(int id)
{
    Product item = repository.Get(id);  // get object from repository
    if (item == null)
    {
        var resp = new HttpResponseMessage(HttpStatusCode.NotFound)
        {
           Content=new StringContent($"No product with ID = {id}"),
           ReasonPhrase = "Product ID Not Found"
        };

        throw new HttpResponseException(resp);
    }
    return item;
}
```

HttpResponseMessage class

- It represents a response to be sent to client with response content and status code

```
HttpResponseMessage()
HttpResponseMessage(HttpStatusCode)
```

Property	Meaning
Content	Gets or sets the content of a HTTP response message.
Headers	Gets the collection of HTTP response headers.
IsSuccessStatusCode	Gets a value that indicates if the HTTP response was successful.
ReasonPhrase	Gets or sets the reason phrase which typically is sent by servers together with the status code.
RequestMessage	Gets or sets the request message which led to this response message.
StatusCode	Gets or sets the status code of the HTTP response.
Version	Gets or sets the HTTP message version.