



# Building an ASP.NET Web Api RESTful service

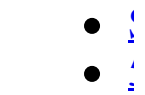
 Posted by **Zoran Maksimovic** on September 19, 2012

25

Tweet

0

Buffer



 Like

14

3

 +1

5

 Share



0



AdChoices

► Visual Data

► XML

► Tutorial

► Example

In this short tutorial I will show how to create quickly a RESTful service by using Visual Studio 2012, ASP.NET Web API and Advanced REST Client.

Download a fully working example that covers all the code mentioned in the post:  
**[FirstAspNetWebApi\\_20120920.zip](#)**

If you liked the article, please share this page to your **twitter** followers  **Tweet This** .

Let’s quickly learn some basic concepts:

## RESTful service

REST acronym stands for **RE**presentational **State** **T**ransfer. In the latest years REST is emerging as a predominant Web service design model. AS another alternative we have Web Services built around SOAP.

RESTful services are fully built around the HTTP protocol, and use the basic HTTP commands like: GET, POST, PUT, DELETE in order to define the “action”. In simple terms, a REST Web Service is all about a **Resource** that can be accessed or changed by using the above mentioned HTTP methods.

What do those HTTP commands represent:

<b>GET</b>	<b>Retrieves</b> a Resource
------------	-----------------------------

### Related posts

- **ASP.NET Web Service - Returning properly formatted error message**
- **What technology for building Web Services in Microsoft.NET?**
- **REST Web Services with Unity.WCF**
- **Fix the WCF wsdl domain name when using IIS6 and HTTPS**
- **Syncfusion: ASP.NET MVC 4 Mobile Web Sites Succinctly**
- **linktotweet.com - open your blog to twitter**
- **ServiceStack: IoC with Microsoft Unity**
- **Import DB-IP.com database data to Microsoft Sql Server with ZORAN.DB.IP.Importer tool**
- **Using the Google DataTable .Net Wrapper**

### Subscribe today!

Subscribe

### Popular articles

- **Building an ASP.NET Web Api RESTful service**
- **ASP.NET Web Service – Returning properly formatted error message**
- **Choose your SQL Server schema comparison tool**
- **Entity Framework Code First – Applying Global Filters**
- **Entity Framework Code First – Filtering And Sorting with paging (1)**

<b>POST</b>	<ul style="list-style-type: none"> <li>• <b>Updates</b> a Resource: if you are requesting the server to update one or more subordinates of the specified resource.</li> <li>• <b>Creates</b> Resource = POST if you are sending a command to the server to create a subordinate of the specified resource, using some server-side algorithm.</li> </ul>
<b>PUT</b>	<ul style="list-style-type: none"> <li>• <b>Creates</b> a Resource. Use PUT if you are sending the full content of the specified resource (URL).</li> <li>• <b>Update</b> a Resource = PUT iff you are updating the full content of the specified resource.</li> </ul>
<b>DELETE</b>	<b>Deletes</b> a Resource.

## Idempotence

**Idempotency** guarantees that repeated invocations of a service capability are safe and will have no negative effect. The side-effects of N > 0 identical requests is the same as for a single request. The methods GET, PUT and DELETE share this property.

## Message Formatting

Usually the main output formats of a restful service are JSON and XML, but obviously this is not the only option as we theoretically could return whatever type of message format we want. For instance, returning a PDF document when doing GET would be absolutely valid.

## SOAP based Web Services

Another way of creating web service is by using SOAP message format as the communication between the Client and Server. Both, SOAP based and RESTful services, even though they try to solve the same issues (retrieving and getting data), the approach is very different. Choosing REST or SOAP based services will depend upon your needs for Security, Speed, Protocols, Extensibility, Legacy Systems, etc. Is not that easy to give a recipe, but in general REST is a great tool when it comes to the AJAX dynamic pages and interoperability as it uses widely known HTTP protocol.

## What is: ASP.NET Web Api

ASP.NET Web API is the latest Microsoft framework that makes it easy to build HTTP services that reach a broad range of clients, including browsers and mobile devices. ASP.NET Web API is an ideal platform for building RESTful applications on the .NET Framework.

ASP.NET Web API is build around (or better say, into) the newest ASP.NET MVC 4 framework and for the time being is the most powerful Microsoft tool when it comes to creating RESTful services. Yes, there are other possibilities as described in the article [What technology for building web services in Microsoft.Net](#).

After this short introduction of very basic concepts, lets do some code.

## ASP.NET Web API Installation

There are few ways of installing ASP.NET Web API.

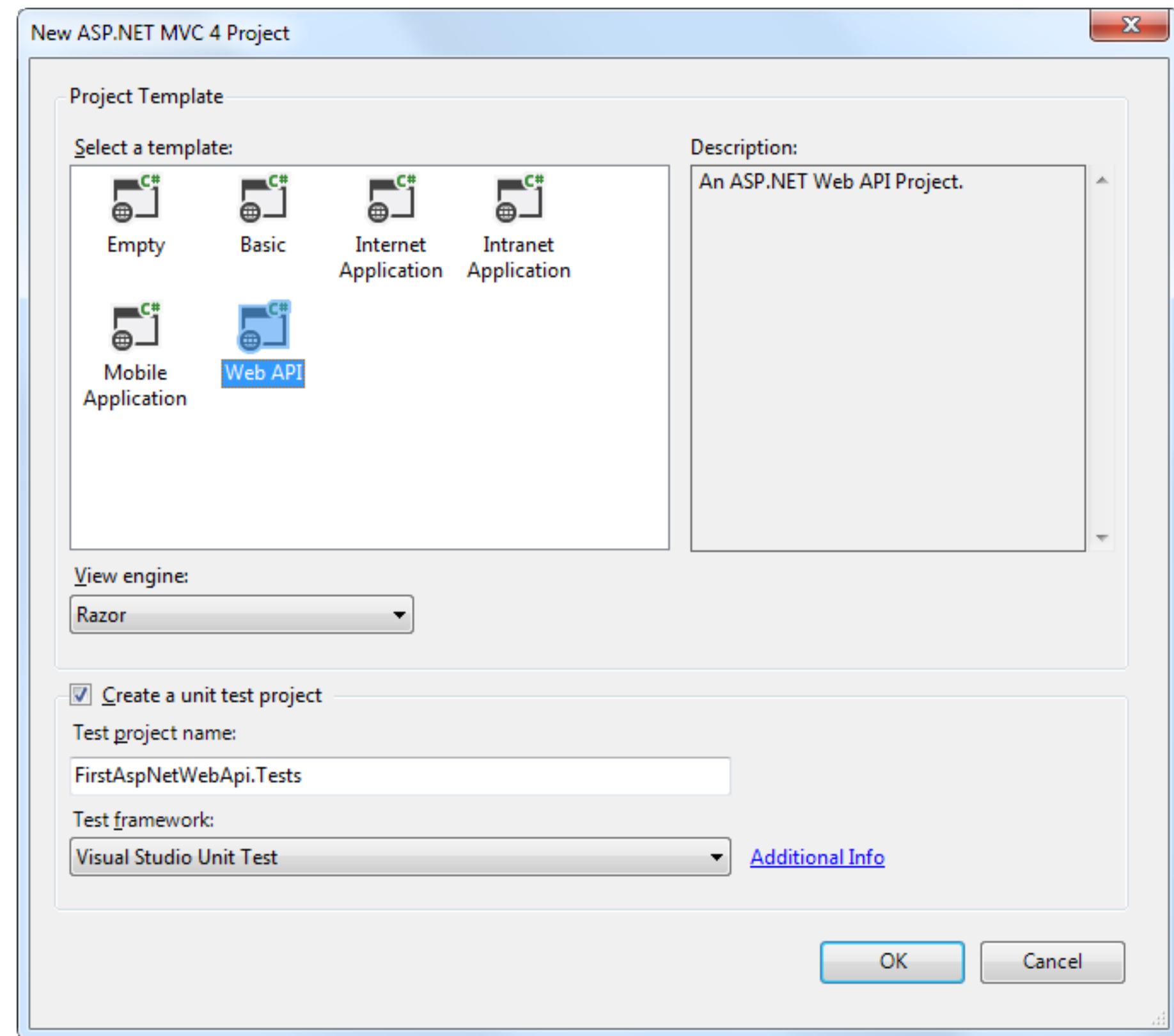
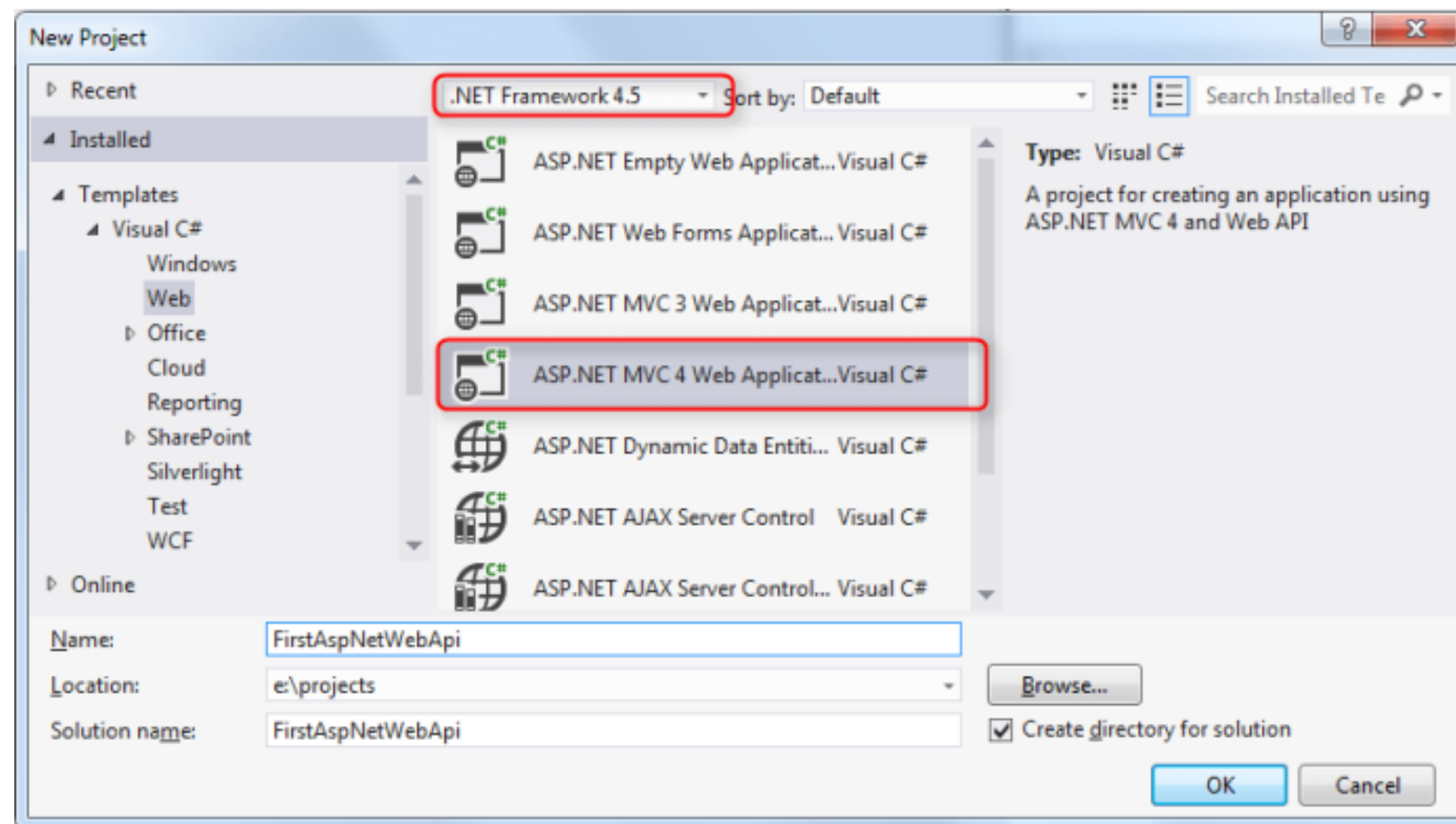
1. If you are using Visual Studio 2012, ASP.NET Web API is already pre-installed, and there is anything else needed to do.
2. If you are using Visual Studio 2010, then there are few possibilities:
  - Use NuGet and by choosing the [Microsoft.AspNet.WebApi](#) package all of the assemblies necessary to run ASP.NET Web API will be installed automatically for you.
  - Choose to download directly the framework from <http://www.asp.net/downloads>
  - Choose to use [Microsoft Web Platform Installer](#) and choose the **ASP.NET MVC 4 Tools update with Language Pack (August 2012)**

## First steps

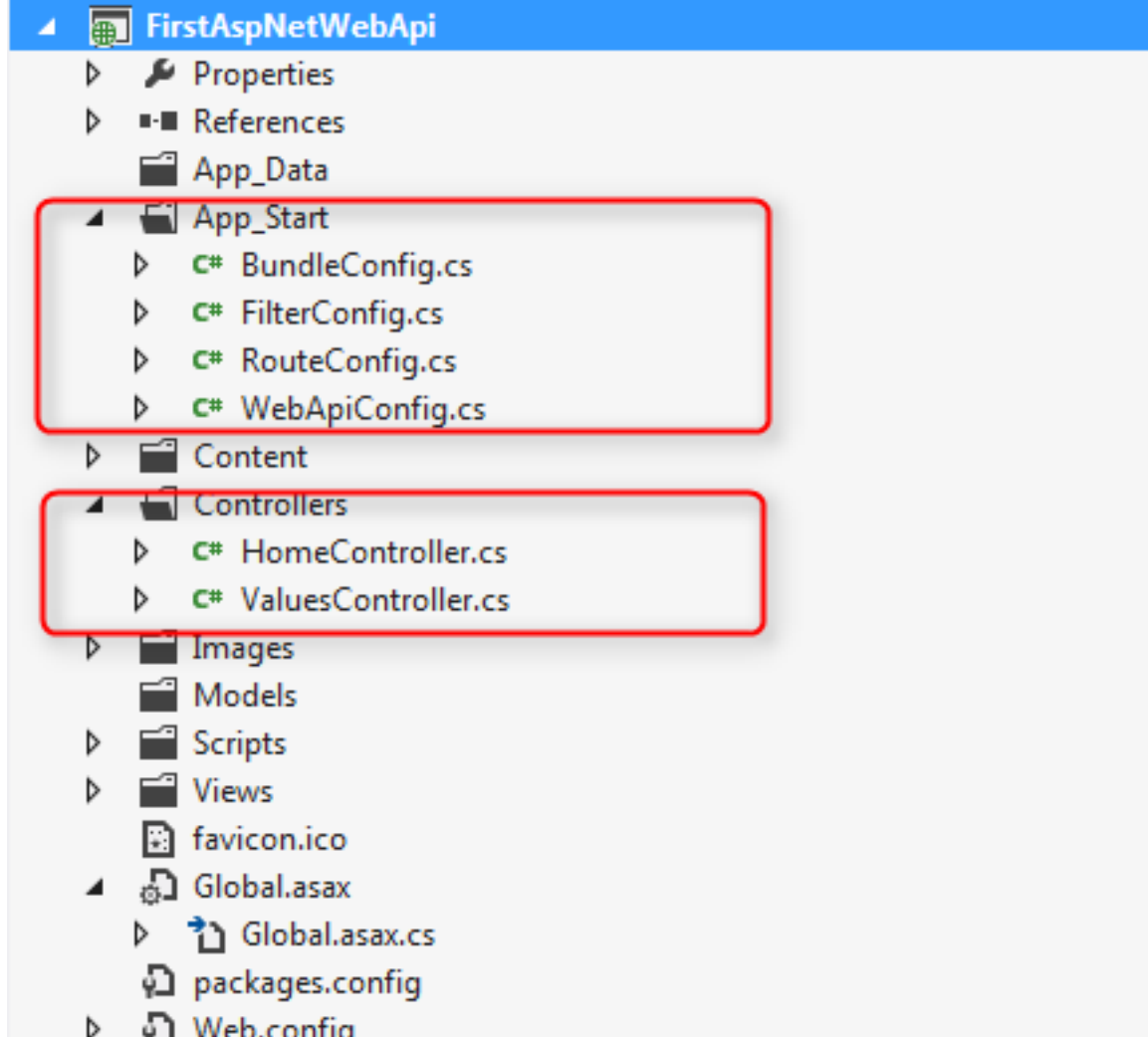
As mentioned, ASP.NET Web API is part of the ASP.NET MVC 4 Framework, and as a such in order to

- [Using the Google DataTable .Net Wrapper](#)
- [Agile software development tools and techniques – introduction](#)
- [Microsoft.NET O/R mapper: choose your own!](#)
- [Embedding SVN revision number at compile time with MsBuild](#)
- [Easy task estimation with Three-point estimation technique](#)

start, we need to create a new project by choosing ASP.NET MVC 4 Application project, and then choose the Web API as shown in the picture below:



Visual Studio will create a basic service structure, and I would like to highlight two folders:



We may easily delete the `HomeController` as we are not interested into creating any MVC application in this example.

## Routing

`WebApiConfig.cs` file in the `App_Start` folder is particularly interesting. It contains the definition of the routing needed for our service to work. As we normally do for the ASP.NET MVC application, we need to define the “route”, or better say the PATH to our service actions.

The code generated by default.

```
1 public static class WebApiConfig
2 {
3     public static void Register(HttpConfiguration config)
4     {
5         config.Routes.MapHttpRoute(
6             name: "DefaultApi",
7             routeTemplate: "api/{controller}/{id}",
8             defaults: new { id = RouteParameter.Optional }
9         );
10    }
11 }
```

Please note the following:

- **routeTemplate** starts with “api/”. This will generate urls like: `www.agile-code.com/api/product/1`. Use “api” keyword to differentiate it from the normal ASP.NET MVC route.
- **{controller}** represents the actual controller. Controller in our case would correspond to the part in bold: `www.agile-code.com/api/product/1`
- There is no “**{action}**” as we would expect it in ASP.NET MVC. As there are no actions but verbs in form of `Post()`, `Get()` ... methods, there is no need for defining the Action in the route. The “{action}” is automatically known at runtime based on the HTTP verb used in the request.

## RESTful service basic code

Visual Studio will take care of creating the basic skeleton of the service itself, as follows:

```
1 public class ValuesController : ApiController
2 {
3     // GET api/values
4     public IEnumerable<string> Get()
5     {
6         return new string[] { "value1", "value2" };
7     }
8
9     // GET api/values/5
10    public string Get(int id)
11    {
12        return "value";
13    }
```



```

14
15 // POST api/values
16 public void Post([FromBody]string value)
17 {
18 }
19
20 // PUT api/values/5
21 public void Put(int id, [FromBody]string value)
22 {
23 }
24
25 // DELETE api/values/5
26 public void Delete(int id)
27 {
28 }
29 }

```

In the code above, there are few things to note:

1. **ApiController** is a new controller that comes as part of the ASP.NET Web API. Every RESTful service should inherit from it.
2. **Methods** in the service itself have the names as mentioned above as the HTTP methods, and those are bound to the proper verb at runtime by the Web API framework. It is still allowed to have methods that contain something else than the basic HTTP methods mentioned before.

## Implementing something more realistic

Lets implement a ProductController, which will represent the Product resource, that for the sake of simplicity will look like:

```

1 public class Product
2 {
3     public int Id { get; set; }
4     public string Name { get; set; }
5 }

```

## Service Implementation

I will show one by one the methods implementing various HTTP verbs.

Important: you will see that I am using repository keyword. This is simply a class that will keep in memory the list of Products, just for the sake of this example.

## GET

### GET a list of products

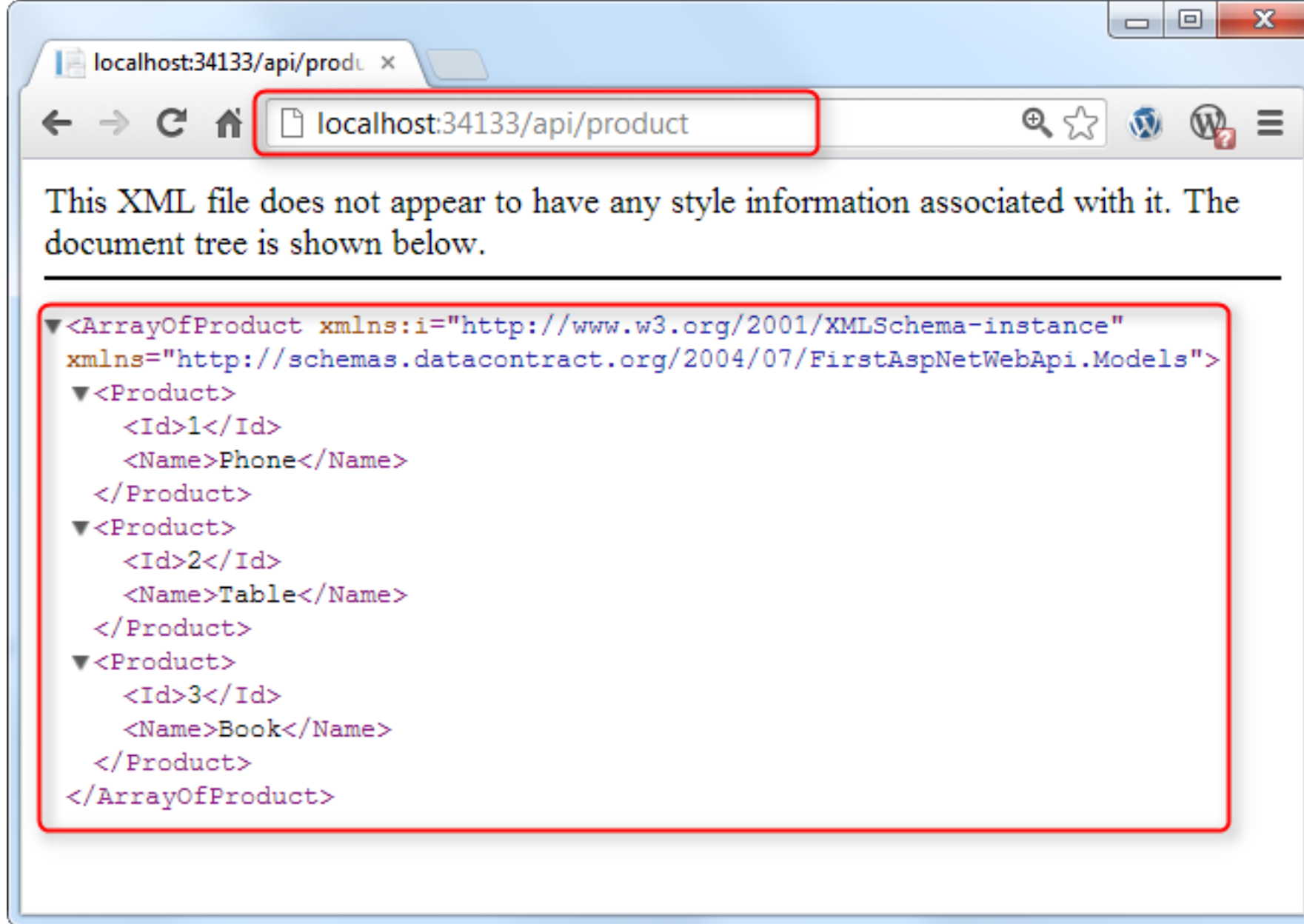
Get is pretty much straightforward. The Method called Get() is defined, and in our case returns a list of Products

```

1 // GET api/product
2 public IEnumerable<Product> Get()
3 {
4     return repository.GetProducts();
5 }

```

for Getting back the data we may simply use the browser as well. So, if we navigate the page (in my case is <http://localhost:34133/api/product>), I will get back the list of products.



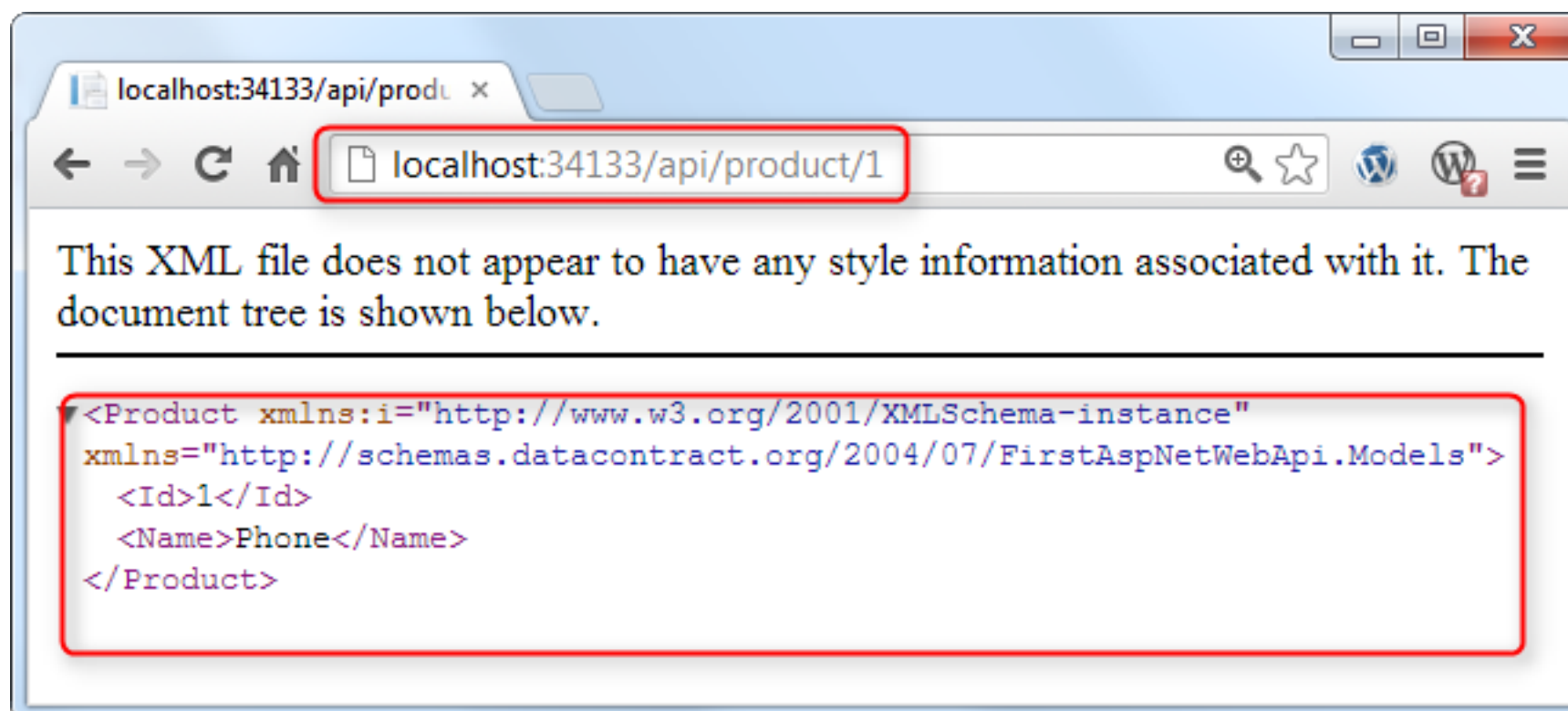
## GET a single product

This is the code that will return one single product:

```
1 // GET api/product/5
2 public Product Get(int id)
3 {
4     Product product = repository.GetProduct(id);
5
6     if (product == null)
7     {
8         throw new HttpResponseException(
9             Request.CreateResponse(HttpStatusCode.NotFound));
10    }
11    else
12    {
13        return product;
14    }
15 }
```

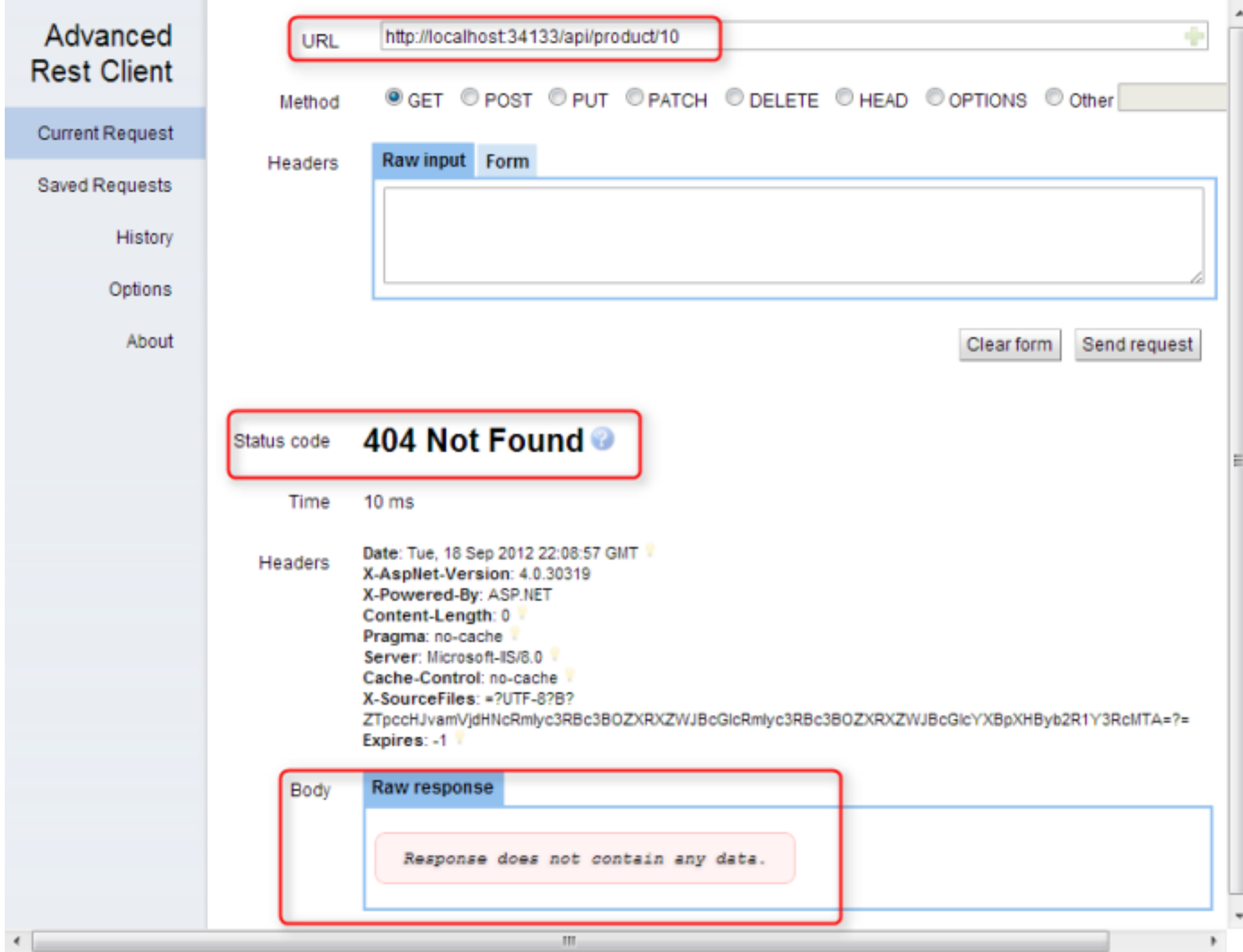
The important bit in this case is that if the PRODUCT id is not found, we are going to throw an error of type `HttpResponseException`. HTTP Response in this case will be "Not Found" message.

This is what happens when we are searching for an existing product:



and in case the product doesn't exist (for this we will use a debugging tool called **Advanced Rest Client** which runs in the Chrome browser in order to test the output.).

As you see the status returned is 404 Not found, with the empty content.



## POST

In order to create new content, we are going to use the POST mechanism to send the data to the server.

This is the code in the service that would create a new Product

```
1 // POST api/product
2 public HttpResponseMessage Post(Product product)
3 {
4     if (ModelState.IsValid)
5     {
6         // this will set the ID for instance...
7         product = repository.AddProduct(product);
8
9         var response = Request.CreateResponse(
10             HttpStatusCode.Created, product);
11
12         string uri = Url.Link("DefaultApi", new {id = product.Id});
13         response.Headers.Location = new Uri(uri);
14         return response;
15     }
16     else
17     {
18         return Request.CreateResponse(HttpStatusCode.BadRequest);
19     }
20 }
```

Creating a new resource involves few steps:

1. The client should send a POST request to api/product to create the resource.
2. REST Service should then return a 201 Created HTTP response, with the URI of the newly created resource in the **Location** header.
3. In case of errors “Bad Request” will be returned to the client.

as shown in the image below, the status HTTP 201 Created together with the actual data is returned to the client application.

URL

Method ☐ GET ☒ POST ☐ PUT ☐ PATCH ☐ DELETE ☐ HEAD ☐ OPTIONS ☐ Other

Headers ☒ Raw input ☐ Form

Body ☒ Raw input ☐ Form ☐ File

Content-type  Content-Type header will overwrite this value.

Status code **201 Created**

Time 54 ms

Headers

Date: Tue, 18 Sep 2012 22:21:13 GMT  
X-AspNet-Version: 4.0.30319  
X-Powered-By: ASP.NET  
Content-Length: 26  
Pragma: no-cache  
Server: Microsoft-IIS/8.0  
Content-Type: application/json; charset=utf-8  
Location: <http://localhost:34133/api/product/6>  
Cache-Control: no-cache  
X-SourceFiles: =?UTF-8?B?ZTpccHJvamVjdHNCcmlyc3RBc3BOZXRXZWJBcGlicmlyc3RBc3BOZXRXZWJBcGlicXBpXHByb2R1Y3Q=?=  
Expires: -1

Body ☒ Raw response ☐ JSON

```
{
  "Id": 6,
  "Name": "Computer"
}
```

Let's check if the product has been actually created by using the browser:

localhost:34133/api/product

localhost:34133/api/product/

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<ArrayOfProduct xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://schemas.datacontract.org/2004/07/FirstAspNetWebApi.Models">
  <Product>
    <Id>1</Id>
    <Name>Phone</Name>
  </Product>
  <Product>
    <Id>2</Id>
    <Name>Table</Name>
  </Product>
  <Product>
    <Id>3</Id>
    <Name>Book</Name>
  </Product>
  <Product>
    <Id>6</Id>
    <Name>Computer</Name>
  </Product>
</ArrayOfProduct>
```

as you may see, the Product with the id 6 is created.

PUT



In order to update an existing product we are going to use the PUT method. Code in the service is defined as follows:

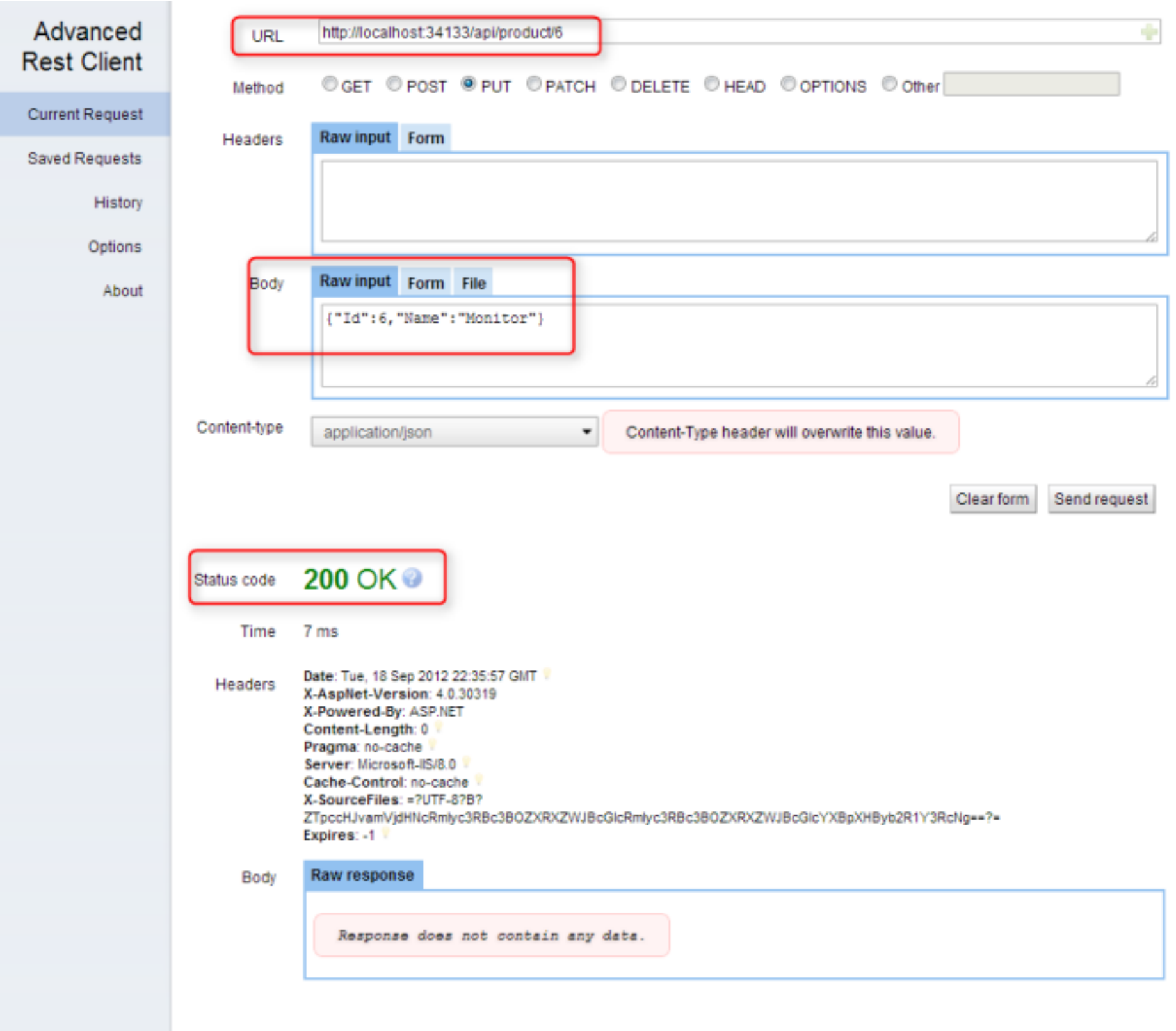
```
1 // PUT api/product/5
2 public HttpResponseMessage Put(int id, Product product)
3 {
4     if (ModelState.IsValid && id == product.Id)
5     {
6         var result = repository.Update(id, product);
7         if (result == false)
8         {
9             return Request.CreateResponse(HttpStatusCode.NotFound);
10        }
11        return Request.CreateResponse(HttpStatusCode.OK);
12    }
13    else
14    {
15        return Request.CreateResponse(HttpStatusCode.BadRequest);
16    }
17 }
```

Note the following:

- 1. There are two parameters. One is the ID of the product we are going to update, and the second parameter is the actual object itself.
- 2. If we try to update a non existing object, the HTTP code `Not Found` will be returned to the client
- 3. If there is any error happening to the code we return `Bad Request`
- 4. If all goes well the HTTP status returned is `200 OK`
- 5. The return parameter is of type `HttpResponseMessage`. We are only returning the status not the content.
- 6. Is not needed to return any Location as in the previous example as know it already. The same comes with the content.

Let’s see this visually:

Posting an update to an existing product will look like this. (Note that the Name of the product changed to be “Monitor”).



in case we try to update a non-existing product. Notice in the address bar, we are trying to update a

product with the id = 11 that doesn't exists. In that case we are returning HTTP status of 404 Not found.

Advanced Rest Client

Current Request

Saved Requests

History

Options

About

URL

http://localhost:34133/api/product/11

Method

GET

POST

PUT

PATCH

DELETE

HEAD

OPTIONS

Other

Headers

Raw input

Form

Body

Raw input

Form

File

{"Id":11,"Name":"Monitor"}

Content-type

application/json

Content-Type header will overwrite this value.

Clear form

Send request

Status code

404 Not Found

Time

8562 ms

Headers

Date: Tue, 18 Sep 2012 22:45:04 GMT

X-AspNet-Version: 4.0.30319

X-Powered-By: ASP.NET

Content-Length: 0

Pragma: no-cache

Server: Microsoft-IIS/8.0

Cache-Control: no-cache

X-SourceFiles: =?UTF-8?B?ZTpccHJvamVjdHNcRmlyc3RBc3BOZXRXZWJ8GlcRmlyc3RBc3BOZXRXZWJ8GlcYXBpXHByb2R1Y3RcMTU=?

Expires: -1

Body

Raw response

Response does not contain any data.

## DELETE

To delete an already existing object we need to pass only the actual ID, as defined in the code below:

```
1 // DELETE api/product/5
2 public HttpResponseMessage Delete(int id)
3 {
4     Product product = repository.GetProduct(id);
5
6     if (product == null)
7     {
8         return Request.CreateResponse(HttpStatusCode.NotFound);
9     }
10
11     try
12     {
13         repository.Delete(id);
14     }
15     catch (Exception exc)
16     {
17         return Request.CreateResponse(HttpStatusCode.BadRequest);
18     }
19
20     return Request.CreateResponse(HttpStatusCode.OK, product);
21 }
```

Note the following:

1. If product we want to delete is not there we return 404 Not found
2. If there is an exception we raise Bad Request
3. In case everything goes well, we return the 200 OK together with the object data, as the client would potentially need to display some information about the product itself.

Product deleted and the status returned as 200 OK.

Advanced Rest Client

Current Request

Saved Requests

History

Options

About

URL

http://localhost:34133/api/product/1

Method

GET

POST

PUT

PATCH

DELETE

HEAD

OPTIONS

Other

Headers

Raw input

Form

Body

Raw input

Form

File

Content-type

application/json

Content-Type header will overwrite this value.

Clear form

Send request

Status code

200 OK

Time

7973 ms

Headers

Date: Tue, 18 Sep 2012 22:50:21 GMT

X-AspNet-Version: 4.0.30319

X-Powered-By: ASP.NET

Content-Length: 23

Pragma: no-cache

Server: Microsoft-IIS/8.0

Content-Type: application/json; charset=utf-8

Cache-Control: no-cache

X-SourceFiles: /\*?UTF-8?B?ZTpccHJvamVjdHNcRmlyc3RBc3BOZXRXZWJBcGlicRmlyc3RBc3BOZXRXZWJBcGlicXBpXHByb2R1Y3RcMQ==?\*

Expires: -1

Body

Raw response

JSON

{

"id": 1,

"name": "Phone"

}

When trying to delete a non existing resource the status 404 Not Found is returned.

Advanced Rest Client

Current Request

Saved Requests

History

Options

About

URL

http://localhost:34133/api/product/122

Method

GET

POST

PUT

PATCH

DELETE

HEAD

OPTIONS

Other

Headers

Raw input

Form

Body

Raw input

Form

File

Content-type

application/json

Content-Type header will overwrite this value.

Clear form

Send request

Status code

404 Not Found

Time

8255 ms

Headers

Date: Tue, 18 Sep 2012 22:58:23 GMT

X-AspNet-Version: 4.0.30319

X-Powered-By: ASP.NET

Content-Length: 0

Pragma: no-cache

Server: Microsoft-IIS/8.0

Cache-Control: no-cache

X-SourceFiles: /\*?UTF-8?B?ZTpccHJvamVjdHNcRmlyc3RBc3BOZXRXZWJBcGlicRmlyc3RBc3BOZXRXZWJBcGlicXBpXHByb2R1Y3RcMTIy?\*

Expires: -1

Body

Raw response

Response does not contain any data.

## Complete code

In order to make it easier for you, here is the full source code of the service.

```
1 public class ProductController : ApiController
2 {
3     private readonly ProductContext repository = new ProductContext();
4
5     // GET api/values
6     public IEnumerable<Product> Get()
7     {
8         return repository.GetProducts();
9     }
10
11    // GET api/product/5
12    public Product Get(int id)
13    {
14        Product product = repository.GetProduct(id);
15
16        if (product == null)
17        {
18            throw new HttpResponseException(
19                Request.CreateResponse(HttpStatusCode.NotFound));
20        }
21        else
22        {
23            return product;
24        }
25    }
26
27    // POST api/product
28    public HttpResponseMessage Post(Product product)
29    {
30        if (ModelState.IsValid)
31        {
32            // this will set the ID for instance...
33            product = repository.AddProduct(product);
34
35            var response = Request.CreateResponse(
36                HttpStatusCode.Created, product);
37
38            string uri = Url.Link("DefaultApi", new {id = product.Id});
39            response.Headers.Location = new Uri(uri);
40            return response;
41        }
42        else
43        {
44            return Request.CreateResponse(HttpStatusCode.BadRequest);
45        }
46    }
47
48    // PUT api/product/5
49    public HttpResponseMessage Put(int id, Product product)
50    {
51        if (ModelState.IsValid && id == product.Id)
52        {
53            var result = repository.Update(id, product);
54            if (result == false)
55            {
56                return Request.CreateResponse(HttpStatusCode.NotFound);
57            }
58            return Request.CreateResponse(HttpStatusCode.OK);
59        }
60        else
61        {
62            return Request.CreateResponse(HttpStatusCode.BadRequest);
63        }
64    }
65
66    // DELETE api/product/5
67    public HttpResponseMessage Delete(int id)
68    {
69        Product product = repository.GetProduct(id);
70
71        if (product == null)
72        {
73            return Request.CreateResponse(HttpStatusCode.NotFound);
74        }
75
76        try
77        {
78            repository.Delete(id);
79        }
80        catch (Exception exc)
81        {
82            return Request.CreateResponse(HttpStatusCode.BadRequest);
83        }
84
85        return Request.CreateResponse(HttpStatusCode.OK, product);
86    }
87 }
```



# Conclusion

ASP.NET Web API is a very versatile framework that allow us to create RESTful services in a very easy way without going through much pain when compared to the WCF configuration part for example. The simplicity of HTTP statuses makes the development even more simple.

## Subscribe today!

Email Address

Subscribe



### About the author:

My name is **Zoran Maksimovic**. I'm a passionate programmer and interested in everything about Software Development, Object-Oriented Design and Software Architecture. Feel free to **contact me** or to know more about me in the **about** section.



Google Shopping Express

 [www.google.com/shopping/express](http://www.google.com/shopping/express)

Unlimited same-day delivery. Free six month membership.



 Twitter 25 25 25

 LinkedIn 5 5 5

 Facebook 14 14 14

 StumbleUpon

 Google +1

 More

Like this:

★ Like

Be the first to like this.

 Category: **Programming**

 Add comments

 Tagged with: **ASP.NET**, **ASP.NET Web API**, **HTTP**, **Kayak**, **Miscrosoft.NET**, **Money**, **mvc**, **Open Source**, **OpenRasta**, **REST**, **RestCake**, **Web**, **Web Service**

## 30 Responses to “Building an ASP.NET Web Api RESTful service”

1.

**Naye** says:  
July 29, 2013 at 7:51 pm



Hi Zoran,

When i run your project ‘out of the box’ from VS 2012 – get the following error:

“HTTP Error 403.14 – Forbidden” – “A default document is not configured for the requested URL, and directory browsing is not enabled on the server.”

Reply
- Zoran Maksimovic** says:  
July 29, 2013 at 8:40 pm



I tried it out and I got the same error. So, you made me worry, as I wrote this article a while ago 😊

Actually, this is simply because there is no default value in the VS 2012 web service that would point to api/product.

However, simply change the URL in the browser to **http://localhost:34133/api/product** and you will get the result 😊

Cheers,  
Zoran

Reply

2. **Lelala** says:  
July 6, 2013 at 11:02 am



Thanks for that tutorial – reading yours, i found the bug in my own code 😊

Reply

**Zoran Maksimovic** says:  
July 6, 2013 at 4:47 pm



Hi,  
It's a pleasure for me to see this kind of comments. thanks!!  
Zoran

Reply

3. **Zied** says:  
June 19, 2013 at 10:13 am



Hi Zoran, Thanks for the post, i implemented it using Entity framework with an SQL database and it works like a shine, i just was wondering how it goes in real world when a user sends data in the post method when i inserted them in the raw input ?  
Thanks

Reply

**Zoran Maksimovic** says:  
June 20, 2013 at 7:39 pm



Hi Zied,  
I guess you front end is a Asp.NET or Asp.NET MVC, or in any case it's a web front end.  
In that case you can create a simple form where the user will fill all the information you need, and then you either post the content of the form to your service, or you can prepare your data, before sending them to the server, by using Javascript. I suggest you use JQuery as it is very versatile and useful.  
Here you have some examples of how to do that  
<http://api.jquery.com/jQuery.post/>

I hope it helps.  
If you find a good solution, please post it here 😊 for our knowledge.

Zoran

Reply

**Zied** says:  
June 24, 2013 at 12:06 pm



Hello, in fact the action is very simple, using Rest service in the final application (inetgrating RestSharp) will give users options to add parameters and we will have smthing like this

```
RestRequest request = new RestRequest();
```

```
request.Method = Method.POST;
```

```
request.AddParameter("parametername","Data");
```

can't beleive how i forgot that!  
i have an other question about the Put Method, i tested and it works but  
values doesn't change !! any idea please?  
Thanks

Reply

4.

**Ron Lutsky** says:  
May 24, 2013 at 8:10 pm



Thanks a lot for this article.

Could you please explain why you use POST to create a new product and not PUT as you stated in “Creates a Resource. Use PUT if you are sending the full content of the specified resource (URL).”

Reply

**Zoran Maksimovic** says:  
May 24, 2013 at 11:02 pm



Hi Ron,

Thanks for your comment. Actually the PUT vs POST causes a lot of confusion.

In reality you could use both for Creation and Update, it just depends how you want to use it, based on the URI to which one are you Posting or Putting.

I think that this article contains a good answer to your question:

<http://stackoverflow.com/questions/630453/put-vs-post-in-rest>

Reply

**Zied** says:  
June 19, 2013 at 9:42 am



Hi Zoran,

thanks for this tuto, i followed it and it works like shine with Entity Framework, i just wondering in the post mode how the user can send data in the real world, for my part i created the service and passed the data in the input Raw to test it but when it comes to a simple user how will he pass parameters to Insert them ?

Thanks Again 😊

Reply

5.

**Ron Lutsky** says:  
May 24, 2013 at 5:23 pm



Great.

At the beginning you say that PUT:

“Creates a Resource. Use PUT if you are sending the full content of the specified resource (URL).”

But you use POST to create a product in the code example.

Can you explain this please? Thanks a lot 😊

Reply

6.

**Kirill** says:  
April 30, 2013 at 2:53 pm



Thanks for tutorial. Is it possible to put product ID as attribute, so that xml output was ?

Reply

**Zoran Maksimovic** says:  
May 26, 2013 at 8:25 pm



Hi Kirill,  
I am not sure what do you really mean?

Reply

7. **Manish Jain** says:  
April 29, 2013 at 7:08 pm



Thanks Zoran for taking time to write this wonderful blog!!!

Reply

**Zoran Maksimovic** says:  
May 2, 2013 at 12:29 pm



Thanks for your comment 😊

Reply

8. **vinaymurthy** says:  
April 4, 2013 at 7:43 am



how to achieve security with restful api in .net

Reply

9. **darshan** says:  
March 29, 2013 at 8:38 am



one thing i want to ask..when we post data using Advance Rest Client.where will be the data stored?

I mean the data we have entered in Advance Rest Client ...will be stored in class file or else where..?

Reply

**Zoran Maksimovic** says:  
April 1, 2013 at 10:04 pm



Hi Darshan,  
I am not quite sure if I understood your question. Advanced Rest Client is just a client application that enables to post JSON or XML to a RESTful service, nothing more. So, that data is not “saved” anywhere really but just transmitted to the service. The RESTful service, before receiving the JSON data, through some filters will receive the deserialized JSON/XML object, for which one obviously you would need to have a class.  
If you have perhaps a more specific question I will be happy to try to help you.

Reply

10. **Rambabu** says:  
March 15, 2013 at 2:06 pm



It's really helpful.



Thanks a lot!!!!!!

Reply

**Zoran Maksimovic** says:  
March 16, 2013 at 8:42 pm



hi Rambabu, my pleasure. Tx for commenting.

Reply

11. **Naresh** says:  
March 4, 2013 at 11:30 am



Hi Zoran,

i am getting below error:

An error has occurred.

The ‘ObjectContext`1’ type failed to serialize the response body for content type ‘application/xml; charset=utf-8’.

System.InvalidOperationException

An error has occurred.

The operation cannot be completed because the DbContext has been disposed.

System.InvalidOperationException


at System.Data.Entity.Internal.InternalContext.CheckContextNotDisposed() at  
System.Data.Entity.Internal.LazyInternalContext.InitializeContext() at  
System.Data.Entity.Internal.InternalContext.Initialize() at  
System.Data.Entity.Internal.Linq.InternalQuery`1.GetEnumerator() at  
System.Data.Entity.Infrastructure.DbQuery`1.System.Collections.Generic.IEnumerable.GetEnumerator() at  
at WriteArrayOfLeadToXml(XmlWriterDelegator , Object , XmlObjectSerializerWriteContext ,  
CollectionDataContract ) at  
System.Runtime.Serialization.CollectionDataContract.WriteXmlValue(XmlWriterDelegator  
xmlWriter, Object obj, XmlObjectSerializerWriteContext context) at  
System.Runtime.Serialization.XmlObjectSerializerWriteContext.WriteDataContractValue(DataContract  
dataContract, XmlWriterDelegator xmlWriter, Object obj, RuntimeTypeHandle  
declaredTypeHandle) at  
System.Runtime.Serialization.XmlObjectSerializerWriteContext.SerializeAndVerifyType(DataContract  
dataContract, XmlWriterDelegator xmlWriter, Object obj, Boolean verifyKnownType,  
RuntimeTypeHandle declaredTypeHandle, Type declaredType) at  
System.Runtime.Serialization.XmlObjectSerializerWriteContext.SerializeWithXsiTypeAtTopLevel(D  
dataContract, XmlWriterDelegator xmlWriter, Object obj, RuntimeTypeHandle  
originalDeclaredTypeHandle, Type graphType) at  
System.Runtime.Serialization.DataContractSerializer.InternalWriteObjectContent(XmlWriterDelega  
writer, Object graph, DataContractResolver dataContractResolver) at  
System.Runtime.Serialization.DataContractSerializer.InternalWriteObject(XmlWriterDelegator  
writer, Object graph, DataContractResolver dataContractResolver) at  
System.Runtime.Serialization.XmlObjectSerializer.WriteObjectHandleExceptions(XmlWriterDelega  
writer, Object graph, DataContractResolver dataContractResolver) at  
System.Runtime.Serialization.DataContractSerializer.WriteObject(XmlWriter writer, Object  
graph) at System.Net.Http.Formatting.XmlMediaTypeFormatter.c\_\_DisplayClass7.b\_\_6() at  
System.Threading.Tasks.TaskHelpers.RunSynchronously(Action action, CancellationToken  
token)

Please , correct me , if i miss anything.

Thanks.

Reply

**Zoran Maksimovic** says:  
March 5, 2013 at 12:15 am



Hi Naresh,  
When are you exactly getting this error? By executing the attached project file?

Reply

12.


**Nikunj Dhawan** says:  
February 1, 2013 at 10:19 am



This is very helpful!

Reply

**Zoran Maksimovic** says:  
February 1, 2013 at 8:19 pm



Thank you Nikunj!

Reply

13.

**Dev** says:  
January 27, 2013 at 6:03 pm




Thanks very helpful!

Reply

14.

**slareau** says:  
January 19, 2013 at 11:36 pm



I have downloaded the code and run it. I can enter a uri and run the app and get results but I am unable to get the form in this example. What am I doing wrong?

Reply

**Zoran Maksimovic** says:  
January 22, 2013 at 10:47 am



Hi,  
For testing purposes I have use the “Advanced Rest Client” tool that you may install in your Chrome browser, if you are using Chrome. You may download it for free from here: <https://chrome.google.com/webstore/detail/advanced-rest-client/hgmloofddffdnphfgcellkdfbfbjeloo>  
If you are using Firefox or IE, there are other extensions that you might use. What I particularly like is the “Fiddler” tool, that you may also download from free (<http://www.fiddler2.com/fiddler2/>) , that allows you to do the same (with much more advanced features).

I hope that it helps! Let me know!

Reply

15.

jon says:

November 6, 2012 at 1:16 am



That's exactly the problem I'm having. I'm new to Web API's, loc and DI and I'm having difficulty combining it all into one solution. There's a lot of examples of IoC and DI OR Web API's, but nothing with a solid architecture (i.e. decoupled code) combining them all.


Anyways, thank you so much for your time!

Reply

16.

jon says:

November 5, 2012 at 2:40 am




Would you be able to provide sample code with an actual database please? I've been really struggling with the full concept. Thank you!

Reply

Zoran Maksimovic says:

November 5, 2012 at 11:41 am



Hi Jon,

For reason of brevity of the article, I was only pointing out the changes and what should be done in order to create a service, without actually going deeper into the database connectivity.

If you check the source code (the example project that you may download) there is a class called ProductContext where I am actually adding in memory some example data, just for the purpose of trying out the solution.

Depending on the database and the underlying technology for accessing database you are using the "solution" would potentially change. So there is no one single answer.

As a potential solution to the existing example

1) You may simply add the connection to the database directly in the ProductContext class and query your database from there. Just to see how it works.

But in an real world example I would use some architecturally more interesting patterns like:

- Repository Pattern (each repository would have single responsibility and hold access to the db for a specific object),
- Interface segregation: You don't want to couple your Service with the database logic implementation, so the coupling between objects should be done through interfaces.
- Inversion Of Control : To actually inject the implementation object at run time.

Try checking this forum for further info about [Setting up web api to pull data from a database using Entity Framework](#) in case you are using SQL Server and want to use Entity Framework.

When i have a bit more time, which I never really have 😊 I will try to extend this solution with what said above.

Reply

Leave a Reply

## The Latest

- [linktotweet.com – open your blog to twitter](#)
- [ServiceStack: IoC with Microsoft Unity](#)
- [Debug Decompiled Code within Visual Studio with Telerik’s JustCode](#)
- [Entity Framework Code First – Filtering And Sorting with paging \(2\)](#)
- [SQL Server ghost records – in a nutshell](#)

## Archives

-  [October 2013](#) (1)
-  [August 2013](#) (1)
-  [July 2013](#) (1)
-  [May 2013](#) (3)
-  [March 2013](#) (3)
-  [February 2013](#) (4)
-  [January 2013](#) (4)
-  [December 2012](#) (3)
-  [November 2012](#) (3)
-  [October 2012](#) (4)
-  [September 2012](#) (5)
-  [August 2012](#) (5)
-  [July 2012](#) (2)
-  [June 2012](#) (3)
-  [May 2012](#) (12)
-  [April 2012](#) (7)

## THANK YOU

Thanks for dropping by! Feel free to join the discussion by leaving comments, and stay updated by subscribing to the [RSS feed](#).

