**Web API Versioning Using URI**

**ASP.NET Web API Versioning Using URI**

In this article, I am going to discuss the **Web API Versioning Using URI** with an example. Please read our previous article where we discussed the [**HMAC Authentication in ASP.NET Web API**](https://dotnettutorials.net/lesson/hmac-authentication-web-api/) application with example. As part of this article, we are going to discuss the following pointers in detail.

1. **Why Web API versioning is required?**
2. **Different options are available in Web API to maintain the versioning.**
3. **How to version a Web API using URI’s?**

**Why Web API versioning is required?**

Once you develop and deploy a Web API service then different clients start consuming your Web API services.

As you know, day by day the business grows and once the business grows then the requirement may change, and once the requirement change then you may need to change the services as well, but the important thing you need to keep in mind is that you need to do the changes to the services in such a way that it should not break any existing client applications who already consuming your services.

This is the ideal scenario when the Web API versioning plays an important role. You need to keep the existing services as it is so that the existing client applications will not break, they worked as it is, and you need to develop a new version of the Web API service which will start consuming by the new client applications.

**What are the Different options available in Web API to maintain the versioning?**

The different options that are available to maintain versioning are as follows

1. URI’s
2. Query String
3. Version Header
4. Accept Header
5. Media Type

**How to version a Web API using URI’s?**

In this article, I am going to discuss how to maintain versioning using URIs. In our upcoming articles, I will discuss the rest of the versioning options one by one. So let us understand Web API Versioning using URI’s with an example.

Create one empty ASP.NET Web API application with the name **WebAPIVersioning**. Once you create the application, let’s create the following **EmployeeV1** model within the Models folder.

**EmployeeV1.cs**

**namespace** *WebAPIVersioning.Models*

**{**

**public** **class** EmployeeV1

**{**

**public** **int** EmployeeID **{** **get**; **set**; **}**

**public** string EmployeeName **{** **get**; **set**; **}**

**}**

**}**

Version 1 of the above Employee class (**EmployeeV1**) has just 2 properties (**EmployeeID** & **EmployeeName**).

Let’s create an empty Web API controller with the name **EmployeesV1Controller** within the Controllers folder which will act as our version 1 controller. Once you create the controller, then please copy and paste the following code in it.

**using** *System.Web.Http;*

**using** *WebAPIVersioning.Models;*

**using** *System.Collections.Generic;*

**using** *System.Linq;*

**namespace** *WebAPIVersioning.Controllers*

**{**

**public** **class** EmployeesV1Controller : ApiController

**{**

List**<**EmployeeV1**>** employees = new List**<**EmployeeV1**>()**

**{**

new EmployeeV1**()** **{** EmployeeID = 101, EmployeeName = "Anurag"**}**,

new EmployeeV1**()** **{** EmployeeID = 102, EmployeeName = "Priyanka"**}**,

new EmployeeV1**()** **{** EmployeeID = 103, EmployeeName = "Sambit"**}**,

new EmployeeV1**()** **{** EmployeeID = 104, EmployeeName = "Preety"**}**,

**}**;

**public** IEnumerable**<**EmployeeV1**>** Get**()**

**{**

**return** employees;

**}**

**public** EmployeeV1 Get**(int** id**)**

**{**

**return** employees.FirstOrDefault**(**s =**>** s.EmployeeID == id**)**;

**}**

**}**

**}**

**Finally, modify the WebApiConfig.cs file as shown below.**

**using** *System.Web.Http;*

**namespace** *WebAPIVersioning*

**{**

**public** **static** **class** WebApiConfig

**{**

**public** **static** **void** Register**(**HttpConfiguration config**)**

**{**

config.MapHttpAttributeRoutes**()**;

config.Routes.MapHttpRoute**(**

name: "Version1",

routeTemplate: "api/v1/employees/{id}",

defaults: new **{** id = RouteParameter.Optional, controller = "EmployeesV1" **}**

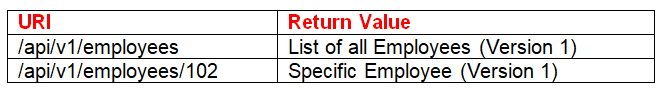
**)**;

**}**

**}**

**}**

The clients of our Version1 Web API service can use the following URLs to get either the list of all employees or a specific employee by using the **EmployeeID**. At the moment, as part of the employee objects, the service returns the **Employee Id** and **Name** properties.



**Implementing Web API Versioning using URI**

Let’s say the business grows and as a result, the requirements have changed and now some of the new clients want the FirstName and LastName properties instead of the Name property. If we change the Version 1 Web API service, then it will break all the existing client applications. So there is a need to create Version 2 of the Web SPI service which will be consumed by the new clients who want the FirstName and LastName properties instead of the Name property.

If we do so, I mean if we create Version 2 of the Web API Service, then all the existing client applications will not break, they work as it is as before and now they have 2 options. If they want they can make use of the version 2 Web API service by making changes to their application or else they still continue their work as it is without changing their application.

So, the important point to keep in mind is that with Web API Versioning we are not breaking any existing client application and at the same time we are also satisfying the new client requirements.

**Following are the steps to create Version 2 of the Web API Service**

**Step1:**Add a class file within the Models folder with the name it **EmployeeV2** and then copy and paste the following code.

**namespace** *WebAPIVersioning.Models*

**{**

**public** **class** EmployeeV2

**{**

**public** **int** EmployeeID **{** **get**; **set**; **}**

**public** string FirstName **{** **get**; **set**; **}**

**public** string LastName **{** **get**; **set**; **}**

**}**

**}**

Notice in **EmployeeV2** class, instead of the Name property, we have the **FirstName** and **LastName** properties.

**Step2:**Add a new Web API 2 empty controller within the Controllers folder with the name “**EmployeesV2Controller**” and then copy and paste the following code.

**using** *System.Collections.Generic;*

**using** *System.Linq;*

**using** *System.Web.Http;*

**using** *WebAPIVersioning.Models;*

**namespace** *WebAPIVersioning.Controllers*

**{**

**public** **class** EmployeesV2Controller : ApiController

**{**

List**<**EmployeeV2**>** employees = new List**<**EmployeeV2**>()**

**{**

new EmployeeV2**()** **{** EmployeeID = 101, FirstName = "Anurag", LastName = "Mohanty"**}**,

new EmployeeV2**()** **{** EmployeeID = 102, FirstName = "Priyanka", LastName = "Dewangan"**}**,

new EmployeeV2**()** **{** EmployeeID = 103, FirstName = "Sambit", LastName = "Satapathy"**}**,

new EmployeeV2**()** **{** EmployeeID = 104, FirstName = "Preety", LastName = "Tiwary"**}**,

**}**;

**public** IEnumerable**<**EmployeeV2**>** Get**()**

**{**

**return** employees;

**}**

**public** EmployeeV2 Get**(int** id**)**

**{**

**return** employees.FirstOrDefault**(**s =**>** s.EmployeeID == id**)**;

**}**

**}**

**}**

Now, the “**EmployeesV2Controller**” returns the “**EmployV2**” object that has the **FirstName** and **LastName** properties instead of the **Name** property.

**Step3:**Modify the **WebApiConfig.cs** file as shown below.

**using** *System.Web.Http;*

**namespace** *WebAPIVersioning*

**{**

**public** **static** **class** WebApiConfig

**{**

**public** **static** **void** Register**(**HttpConfiguration config**)**

**{**

config.MapHttpAttributeRoutes**()**;

config.Routes.MapHttpRoute**(**

name: "Version1",

routeTemplate: "api/v1/employees/{id}",

defaults: new **{** id = RouteParameter.Optional, controller = "EmployeesV1" **}**

**)**;

config.Routes.MapHttpRoute**(**

name: "Version2",

routeTemplate: "api/v2/employees/{id}",

defaults: new **{** id = RouteParameter.Optional, controller = "EmployeesV2" **}**

**)**;

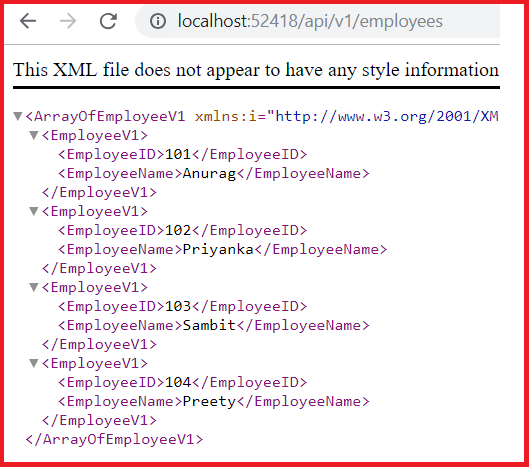
**}**

**}**

**}**

As shown in the above **WebApiConfig.cs** file, now we have 2 routes. Notice the route template, for each of the routes. The  
Version 1 clients use **“/api/v1/employees/{id}”** route while the  
Version 2 clients use **“/api/v2/employees/{id}”** route

That’s it. We have implemented the versioning in our Web API service. So, if we navigate to **/api/v1/employees** – We get the employee with Employee Id and Name properties as shown below



**/api/v2/employees** – We get employees with Employee Id, FirstName and LastName properties as shown below.



At the moment we are using the convention-based routing to implement the Web API versioning. We can also use the Attribute Routing instead of convention-based routing to implement the Web API versioning. What we need to do is, we need to use the [Route] attribute on methods in EmployeesV1Controller and EmployeesV2Controller as shown below.

**EmployeesV1Controller**

**namespace** *WebAPIVersioning.Controllers*

**{**

**public** **class** EmployeesV1Controller : ApiController

**{**

List**<**EmployeeV1**>** employees = new List**<**EmployeeV1**>()**

**{**

new EmployeeV1**()** **{** EmployeeID = 101, EmployeeName = "Anurag"**}**,

new EmployeeV1**()** **{** EmployeeID = 102, EmployeeName = "Priyanka"**}**,

new EmployeeV1**()** **{** EmployeeID = 103, EmployeeName = "Sambit"**}**,

new EmployeeV1**()** **{** EmployeeID = 104, EmployeeName = "Preety"**}**,

**}**;

**[**Route**(**"api/v1/employees"**)]**

**public** IEnumerable**<**EmployeeV1**>** Get**()**

**{**

**return** employees;

**}**

**[**Route**(**"api/v1/employees/{id}"**)]**

**public** EmployeeV1 Get**(int** id**)**

**{**

**return** employees.FirstOrDefault**(**s =**>** s.EmployeeID == id**)**;

**}**

**}**

**}**

**EmployeesV2Controller**

**namespace** *WebAPIVersioning.Controllers*

**{**

**public** **class** EmployeesV2Controller : ApiController

**{**

List**<**EmployeeV2**>** employees = new List**<**EmployeeV2**>()**

**{**

new EmployeeV2**()** **{** EmployeeID = 101, FirstName = "Anurag", LastName = "Mohanty"**}**,

new EmployeeV2**()** **{** EmployeeID = 102, FirstName = "Priyanka", LastName = "Dewangan"**}**,

new EmployeeV2**()** **{** EmployeeID = 103, FirstName = "Sambit", LastName = "Satapathy"**}**,

new EmployeeV2**()** **{** EmployeeID = 104, FirstName = "Preety", LastName = "Tiwary"**}**,

**}**;

**[**Route**(**"api/v2/employees"**)]**

**public** IEnumerable**<**EmployeeV2**>** Get**()**

**{**

**return** employees;

**}**

**[**Route**(**"api/v2/employees/{id}"**)]**

**public** EmployeeV2 Get**(int** id**)**

**{**

**return** employees.FirstOrDefault**(**s =**>** s.EmployeeID == id**)**;

**}**

**}**

**}**

As we are using the Attribute Routing, so we can safely comment following 2 route templates in **WebApiConfig.cs** file

**namespace** *WebAPIVersioning*

**{**

**public** **static** **class** WebApiConfig

**{**

**public** **static** **void** Register**(**HttpConfiguration config**)**

**{**

config.MapHttpAttributeRoutes**()**;

//config.Routes.MapHttpRoute(

// name: "Version1",

// routeTemplate: "api/v1/employees/{id}",

// defaults: new { id = RouteParameter.Optional, controller = "EmployeesV1" }

//);

//config.Routes.MapHttpRoute(

// name: "Version2",

// routeTemplate: "api/v2/employees/{id}",

// defaults: new { id = RouteParameter.Optional, controller = "EmployeesV2" }

//);

**}**

**}**

**}**

At this point, build the solution and test the application and it should work exactly the same way as before.

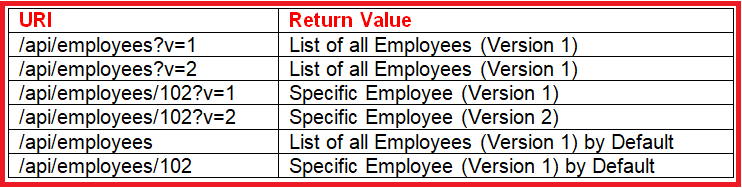
**Web API Versioning using Query string Parameter**

**Web API Versioning using Query String Parameter**

In this article, I am going to discuss **Web API Versioning using the Query String Parameter** with an example. Please read our previous article before proceeding to this article as we are going to work with the same example. In our [**previous article**](https://dotnettutorials.net/lesson/web-api-versioning-using-uri/)**,** we discussed the following things in detail.

1. **Why Web API versioning is required?**
2. **Different options are available in Web API to maintain the versioning.**
3. **How to version a Web API using URI’s?**

In this article, we want to implement versioning a Web API service using Query String Parameter. Let’s have a look at the following table



So when we pass the query string v=1 then we want to return Version 1 of the Web API service and similarly when we pass v=2 then we need to return version 2 of the Web API service. If we don’t specify the query string then by default it will return the version 1 of the Web API Service.

**How the Web API Framework select a Controller?**

Before we start implementing the Web API versioning using a query string parameter, let first understand how the Web API Framework selects a controller when a request is issued to a Web API service. For example, let us understand how the Web API framework selects the controller when a request is issued to the following URI

**/api/employees/1**  
In ASP.NET Web API, there is a class called **DefaultHttpControllerSelector**. This class has a method called **SelectController()** which will select the controller based on the information it has in the URI.

**The above class uses the below algorithm to find the controller:**

1. First, it will look at the route dictionary collection for the key “controller”.
2. Secondly, it takes the value for the “controller” key and appends the string “Controller” to get the controller type name.
3. Finally, it looks for a Web API controller with this type name.

For example, if the route dictionary contains the key-value pair “**controller**” = “**Employees**”, then the controller type is “**EmployeesController**”. If there is no matching type found, or it found multiple matches, then the ASP.NET WEB API Framework simply returns an error to the client.

**In our example, in the URI we have**

1. The name of the controller, in this case, Employees
2. The id parameter value, in this case, 1

So from the URI, the **SelectController()** method takes the name of the controller in this case “**Employees**” and try to find the “**EmployeesController**” in the **route dictionary** and if it founds then returns it. This is the default implementation to select the Web API Controller. If you want to learn more about the controller and action selector in Web API, then please refer to the following article.

[**The Controller and Action Selector in Web API**](https://dotnettutorials.net/lesson/routing-and-action-selection-in-web-api/)

The above default implementation of Controller Selection will not work for us because in our Web API Service we do not have any controller with the name **EmployeesController**. Instead we have the following two controllers

**1. EmployeesV1Controller** and  
2. **EmployeesV2Controller**  
So, when a request is issued to the following URI, depending on the query string parameter “**v**” value we want to select the appropriate controller. If the value is 1, then we need to select the **EmployeesV1Controller**, and if the value is 2, then we need to select the **EmployeesV2Controller**.

**/api/employees?v=1**

**How to Maintain Web API service versioning using a query string parameter?**

**Step1:** As we already discussed, the default implementation of controller selection provided by Web API Framework will not work for us, so we need to provide our own custom controller selector implementation.

To do this, add a new folder to your project with the name “**Custom**” and then add a class file within the **Custom** Folder and name it as “**CustomControllerSelector**“. Once you add the class file, then copy and paste the following code. The code is self-explained so please go through the comments.

**using** *System.Net.Http;*

**using** *System.Web;*

**using** *System.Web.Http;*

**using** *System.Web.Http.Controllers;*

**using** *System.Web.Http.Dispatcher;*

**namespace** *WebAPIVersioning.Custom*

**{**

// Derive the CustomControllerSelector from the DefaultHttpControllerSelector class

**public** **class** CustomControllerSelector : DefaultHttpControllerSelector

**{**

**private** HttpConfiguration \_config;

**public** CustomControllerSelector**(**HttpConfiguration config**)** : **base(**config**)**

**{**

\_config = config;

**}**

**public** **override** HttpControllerDescriptor SelectController**(**HttpRequestMessage request**)**

**{**

// First fetch all the available Web API controllers

var controllers = GetControllerMapping**()**;

// Get the controller name and the parameter values from the request URI

var routeData = request.GetRouteData**()**;

// Get the controller name from route data.

// The name of the controller in our case is "Employees"

var controllerName = routeData.Values**[**"controller"**]**.ToString**()**;

// Set the Default version number to 1

string versionNumber = "1";

var versionQueryString = HttpUtility.ParseQueryString**(**request.RequestUri.Query**)**;

**if** **(**versionQueryString**[**"v"**]** != **null)**

**{**

versionNumber = versionQueryString**[**"v"**]**;

**}**

**if** **(**versionNumber == "1"**)**

**{**

// if the version number is 1, then append V1 to the controller name.

// So at this point the, controller name will become EmployeesV1

controllerName = controllerName + "V1";

**}**

**else**

**{**

// if version number is 2, then append V2 to the controller name.

// So at this point the controller name will become EmployeesV2

controllerName = controllerName + "V2";

**}**

HttpControllerDescriptor controllerDescriptor;

**if** **(**controllers.TryGetValue**(**controllerName, out controllerDescriptor**))**

**{**

**return** controllerDescriptor;

**}**

**return** **null**;

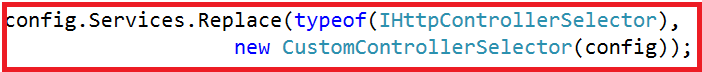
**}**

**}**

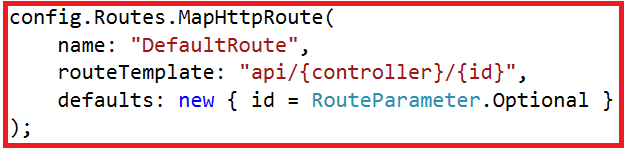
**}**

**Step2:**

Now we need to replace the default controller selector with our custom controller selector and we need to do this within the **WebApiConfig.cs** file. Notice, here we are replacing the **IHttpControllerSelector**, with our **CustomControllerSelector**. The **DefaultHttpControllerSelector** implements **IHttpControllerSelector**, so that is the reason we are replacing **IHttpControllerSelector**.



**Step3:** Include the following default route in WebApiConfig.cs



After the above two changes in the WebApiConfig.cs file, your WebApiConfig.csfile should looks as shown below.

**using** *System.Web.Http;*

**using** *System.Web.Http.Dispatcher;*

**using** *WebAPIVersioning.Custom;*

**namespace** *WebAPIVersioning*

**{**

**public** **static** **class** WebApiConfig

**{**

**public** **static** **void** Register**(**HttpConfiguration config**)**

**{**

config.MapHttpAttributeRoutes**()**;

config.Services.Replace**(**typeof**(**IHttpControllerSelector**)**,

new CustomControllerSelector**(**config**))**;

config.Routes.MapHttpRoute**(**

name: "DefaultRoute",

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

defaults: new **{** id = RouteParameter.Optional **}**

**)**;

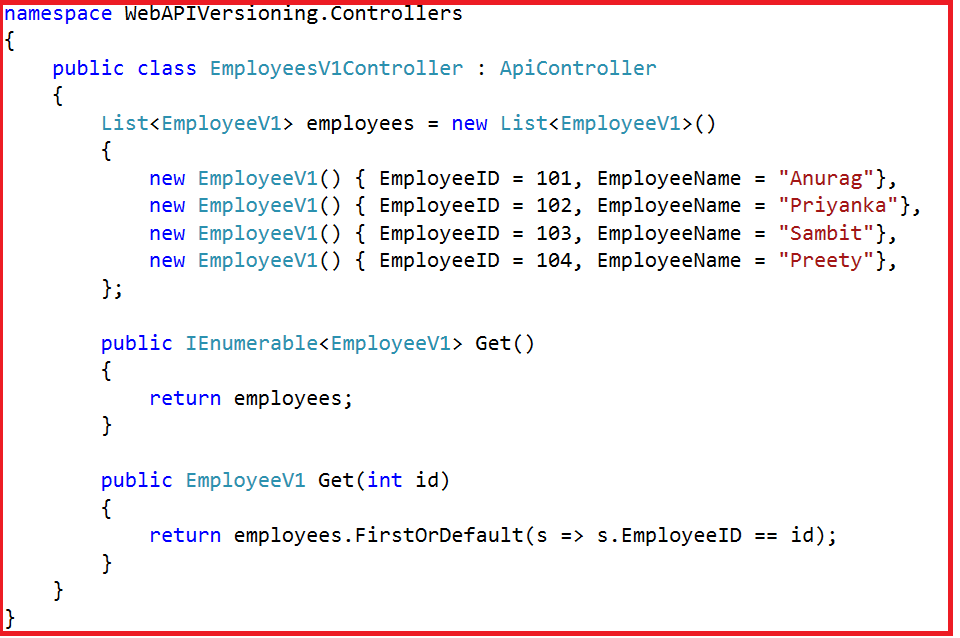
**}**

**}**

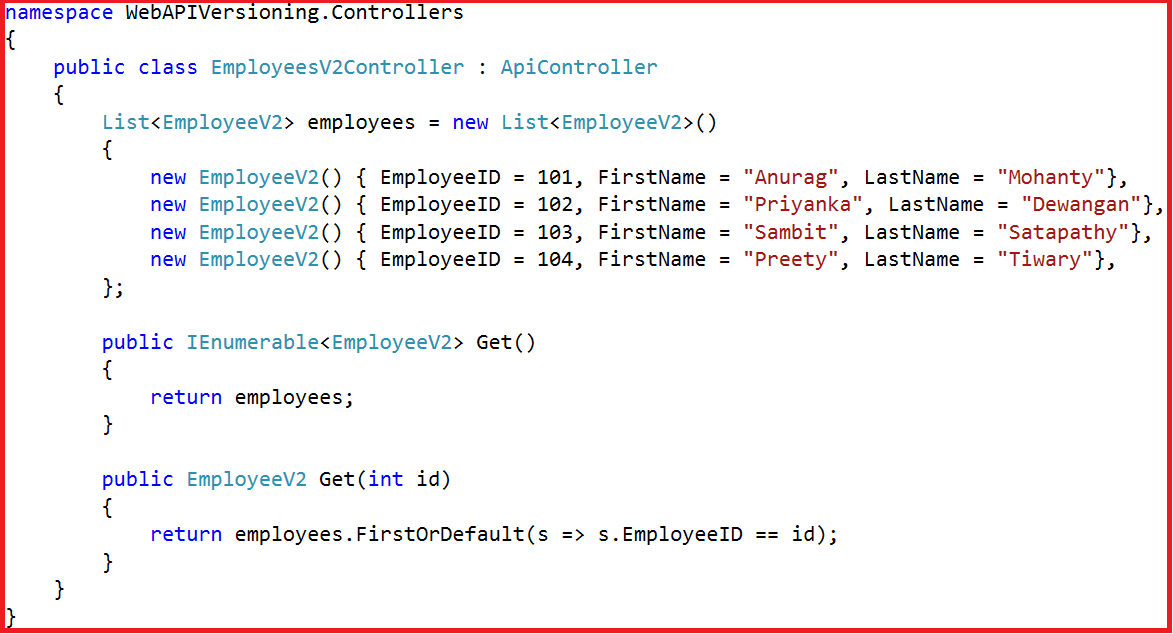
**}**

**Step4:**

Finally remove the **[Route]** attribute, from all the action methods of both **EmployeesV1Controller**and **EmployeesV2Controller**as shown below.



**EmployeesV2Controller**



That’s it; we have done with our implementation. Now run the application and navigates the following URI and you will see the output as expected.

/api/employees  
/api/employees?v=1  
/api/employees?v=2

**Web API Versioning Using Custom Header**

**Web API Versioning Using Custom Header**

In this article, I am going to discuss **Web API Versioning Using Custom Header** with an example. This is a continuation part of our previous article, so please read our previous article before proceeding to this article where we discussed [**Web API Versioning using Query String Parameter**](https://dotnettutorials.net/lesson/web-api-versioning-using-query-string-parameter/).

In our previous article, we implemented a **CustomControllerSelector** which will retrieve the version number from a query string parameter and then based on the version number it will select the appropriate controller.

To implement the Web API Versioning using custom header, all we need to do is to change the logic of the **CustomControllerSelector** class to read the version number from the custom header instead of the query string parameter.

Modify the **CustomControllerSelector** class as shown below. The code is self-explained, so please go through the comments.

**using** *System.Linq;*

**using** *System.Net.Http;*

**using** *System.Web.Http;*

**using** *System.Web.Http.Controllers;*

**using** *System.Web.Http.Dispatcher;*

**namespace** *WebAPIVersioning.Custom*

**{**

// Derive the CustomControllerSelector from the DefaultHttpControllerSelector class

**public** **class** CustomControllerSelector : DefaultHttpControllerSelector

**{**

**private** HttpConfiguration \_config;

**public** CustomControllerSelector**(**HttpConfiguration config**)** : **base(**config**)**

**{**

\_config = config;

**}**

**public** **override** HttpControllerDescriptor SelectController**(**HttpRequestMessage request**)**

**{**

// First fetch all the available Web API controllers

var controllers = GetControllerMapping**()**;

// Get the controller name and the parameter values from the request URI

var routeData = request.GetRouteData**()**;

// Get the controller name from route data.

// The name of the controller in our case is "Employees"

var controllerName = routeData.Values**[**"controller"**]**.ToString**()**;

// Set the Default version number to 1

string versionNumber = "1";

// Get the version number from Custom header

// You can give any name to this custom header. You have to use this

// same header to specify the version number when issuing a request

string customHeader = "X-EmployeesService-Version";

**if** **(**request.Headers.Contains**(**customHeader**))**

**{**

versionNumber = request.Headers.GetValues**(**customHeader**)**.FirstOrDefault**()**;

**}**

**if** **(**versionNumber == "1"**)**

**{**

// if the version number is 1, then append V1 to the controller name.

// So at this point the, controller name will become EmployeesV1

controllerName = controllerName + "V1";

**}**

**else**

**{**

// if version number is 2, then append V2 to the controller name.

// So at this point the controller name will become EmployeesV2

controllerName = controllerName + "V2";

**}**

HttpControllerDescriptor controllerDescriptor;

**if** **(**controllers.TryGetValue**(**controllerName, out controllerDescriptor**))**

**{**

**return** controllerDescriptor;

**}**

**return** **null**;

**}**

**}**

**}**

That’s it; we have done with our implementation. Run the application and issue a request to **/api/employees** using either **Fiddler** or **Postman**. If you are new to Fiddler or Postman, then please read the following two articles where we discussed how to download, install and use Fiddler and Postman to test Web API Services.

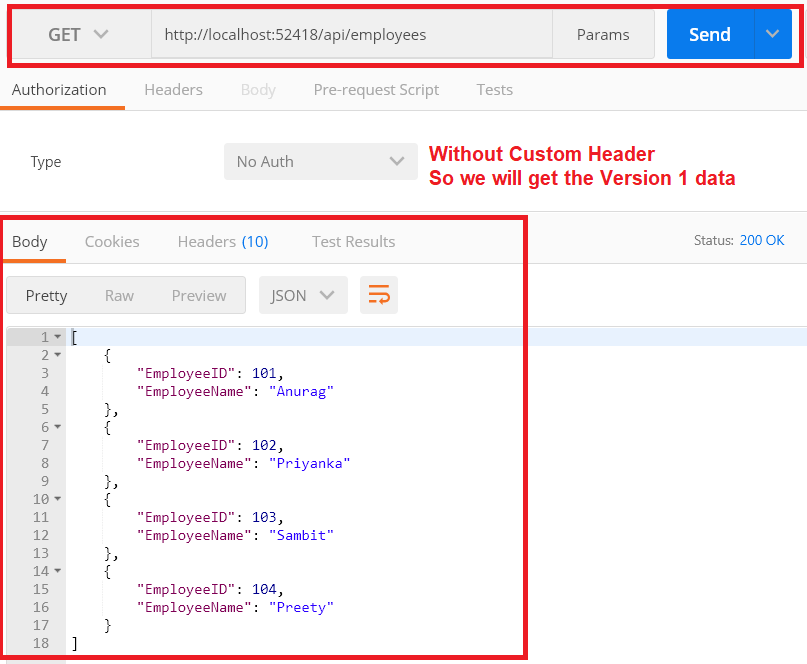
[**Using Fiddler to Test Web API Services**](https://dotnettutorials.net/lesson/how-to-use-fiddler-to-test-web-api/)

[**Using Postman to Test Web API Services**](https://dotnettutorials.net/lesson/how-to-use-postman-to-test-web-api/)

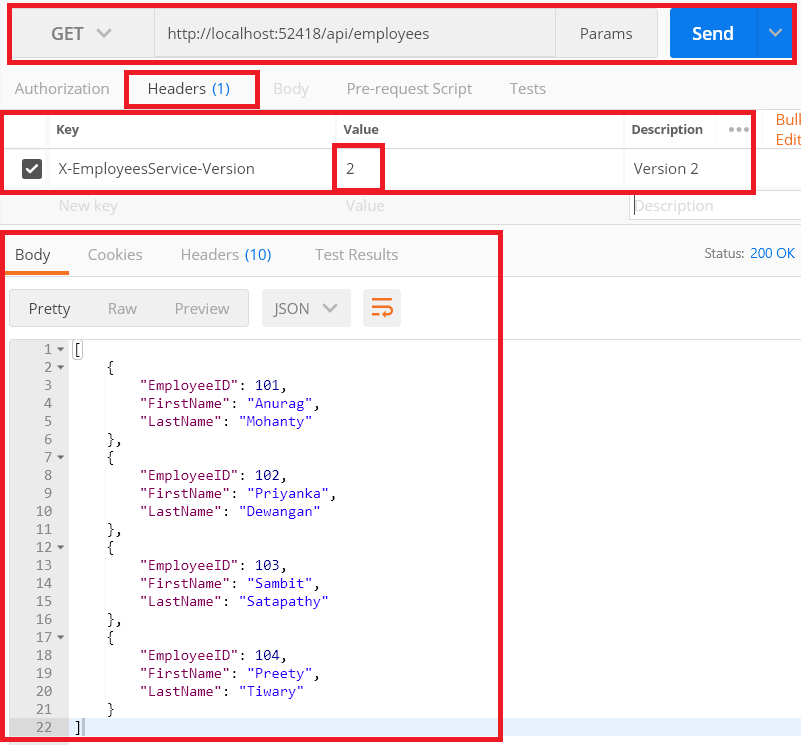
Here we are going to use the Postman to test the service

**Test1:**

In the below request, we have not specified the custom header. So we issue the request, we get back EmployeeV1 objects as the response the reason is we have set version 1 as the default in our code.



**Test2:**Let’s issue another request using the custom header “**X-EmployeesService-Version**” as shown below.



As shown in the above image, we got the version 2 response as expected this is because in the header we pass the value 2 for the custom header **X-EmployeesService-Version**.

**Web API Versioning Using Accept Header**

**Web API Versioning Using Accept Header**

In this article, I am going to discuss **Web API Versioning Using Accept Header** with an example. This is a continuation part of our **previous article**, so please read our previous article before proceeding to this article where we discussed [**Web API Versioning Using Custom Header**](https://dotnettutorials.net/lesson/web-api-versioning-using-custom-header/).

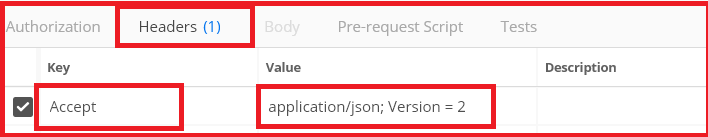
**What is the Accept Header?**

The Accept header tells the server in which file format the browser wants the data. These file formats are generally called as MIME-types. The MIME stands for Multipurpose Internet Mail Extensions.

In the [**Content Negotiation in Web API**](https://dotnettutorials.net/lesson/content-negotiation-web-api/)**article**, we discussed that, if we want the Web API service to return data in XML format then we need to set the Accept header value to **application/xml**. Similarly, if we want the Web API service to return the data in JSON format then we need to set the Accept header value to **application/json**.

**Implementing Web API Versioning using accept header**

In our last article, we discussed how to create a custom header for versioning. So, instead of creating a custom header, which is used only for versioning purposes, we can make use of the built-in Accept header. Within the Accept header, we can add additional parameters to send the additional data along with the request to the server. For example, we can specify the version of the service that we want by using the version parameter as shown in the below image.



What we need to do here is, now we need to read the Version parameter value from the Accept header in our **CustomControllerSelector** class. So, modify the **CustomControllerSelector** class as shown below. The code is self-explained, so please read the comments.

**using** *System.Linq;*

**using** *System.Net.Http;*

**using** *System.Web.Http;*

**using** *System.Web.Http.Controllers;*

**using** *System.Web.Http.Dispatcher;*

**namespace** *WebAPIVersioning.Custom*

**{**

// Derive the CustomControllerSelector from the DefaultHttpControllerSelector class

**public** **class** CustomControllerSelector : DefaultHttpControllerSelector

**{**

**private** HttpConfiguration \_config;

**public** CustomControllerSelector**(**HttpConfiguration config**)** : **base(**config**)**

**{**

\_config = config;

**}**

**public** **override** HttpControllerDescriptor SelectController**(**HttpRequestMessage request**)**

**{**

// First fetch all the available Web API controllers

var controllers = GetControllerMapping**()**;

// Get the controller name and the parameter values from the request URI

var routeData = request.GetRouteData**()**;

// Get the controller name from route data.

// The name of the controller in our case is "Employees"

var controllerName = routeData.Values**[**"controller"**]**.ToString**()**;

// Set the Default version number to 1

string versionNumber = "1";

// Get the version number from the Accept header

// Users can include multiple Accept headers in the request

// Check if any of the Accept headers has a parameter with name version

var acceptHeader = request.Headers.Accept.Where**(**a =**>** a.Parameters

.Count**(**p =**>** p.Name.ToLower**()** == "version"**)** **>** 0**)**;

// If there is atleast one header with a "version" parameter

**if** **(**acceptHeader.Any**())**

**{**

// Get the version parameter value from the Accept header

versionNumber = acceptHeader.First**()**.Parameters

.First**(**p =**>** p.Name.ToLower**()** == "version"**)**.Value;

**}**

**if** **(**versionNumber == "1"**)**

**{**

// if the version number is 1, then append V1 to the controller name.

// So at this point the, controller name will become EmployeesV1

controllerName = controllerName + "V1";

**}**

**else**

**{**

// if version number is 2, then append V2 to the controller name.

// So at this point the controller name will become EmployeesV2

controllerName = controllerName + "V2";

**}**

HttpControllerDescriptor controllerDescriptor;

**if** **(**controllers.TryGetValue**(**controllerName, out controllerDescriptor**))**

**{**

**return** controllerDescriptor;

**}**

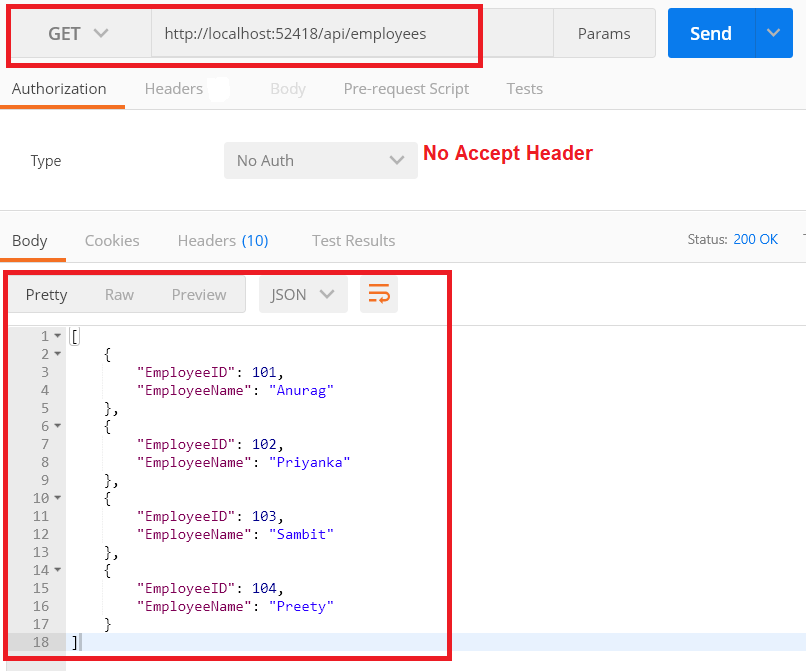
**return** **null**;

**}**

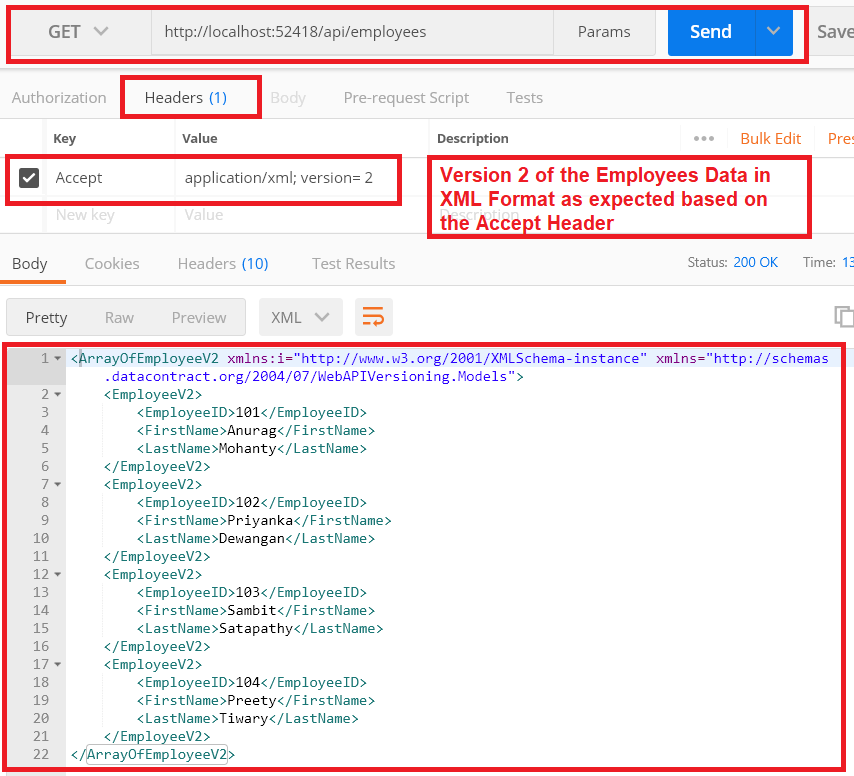
**}**

**}**

That’s it; we have done with our implementation. Run the application and then issue a request without the Accept header using the postman as the client, the service falls back to version 1 and returns version 1 employee objects in JSON format as expected as shown in the below image.



Now, specify the version parameter as part of the Accept header then you will get the response in the specified version as expected as shown in the below image.

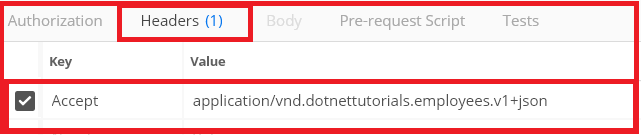


**Web API Versioning Using Custom Media Types**

**Web API Versioning Using Custom Media Types**

In this article, I am going to discuss **Web API Versioning Using Custom Media Types** i.e. Vendor Specific Media Types with an example. This is a continuation part of our **previous article**, so please read our previous article before proceeding to this article where we discussed [**Web API Versioning Using Accept Header**](https://dotnettutorials.net/lesson/web-api-versioning-using-accept-header/).

So what we want to do here is, instead of using the built-in media types such as **application/json** or **application/xml,** we want to use our custom media type which is shown in the below postman request.



Notice that in the media type, we have specified the version of the service that we want. The custom media types are prefixed with **vnd** which indicates that this media type is a vendor-specific media type.

So what we need to do from our **CustomControllerSelector**class is to read the version number from the custom media type that we have specified in the request.

Modify the **CustomControllerSelector** class as shown below where we implement the logic to read the version number from the custom media type. The code is self-explanatory, so please go through the lines.

**CustomControllerSelector**

**using** *System.Linq;*

**using** *System.Net.Http;*

**using** *System.Text.RegularExpressions;*

**using** *System.Web.Http;*

**using** *System.Web.Http.Controllers;*

**using** *System.Web.Http.Dispatcher;*

**namespace** *WebAPIVersioning.Custom*

**{**

// Derive the CustomControllerSelector from the DefaultHttpControllerSelector class

**public** **class** CustomControllerSelector : DefaultHttpControllerSelector

**{**

**private** HttpConfiguration \_config;

**public** CustomControllerSelector**(**HttpConfiguration config**)** : **base(**config**)**

**{**

\_config = config;

**}**

**public** **override** HttpControllerDescriptor SelectController**(**HttpRequestMessage request**)**

**{**

// First fetch all the available Web API controllers

var controllers = GetControllerMapping**()**;

// Get the controller name and the parameter values from the request URI

var routeData = request.GetRouteData**()**;

// Get the controller name from route data.

// The name of the controller in our case is "Employees"

var controllerName = routeData.Values**[**"controller"**]**.ToString**()**;

// Set the Default version number to 1

string versionNumber = "1";

// Get the version number from the Custom media type

// We need to use the regular expression for mataching the pattern of the

// media type. We have given a name for the matched group that contains

// the version number which enables us to retrieve the version number

// using the group name("version") instead of ZERO based index

string regex = @"application\/vnd\.dotnettutorials\.([a-z]+)\.v(?<version>[0-9]+)\+([a-z]+)";

// Users can include multiple Accept headers in the request.

// So we need to check atlest if any of the Accept headers has our custom

// media type by checking if there is a match with regular expression specified

var acceptHeader = request.Headers.Accept

.Where**(**a =**>** Regex.IsMatch**(**a.MediaType, regex, RegexOptions.IgnoreCase**))**;

// If there is atleast one Accept header with our custom media type

**if** **(**acceptHeader.Any**())**

**{**

// Retrieve the first custom media type

var match = Regex.Match**(**acceptHeader.First**()**.MediaType, regex, RegexOptions.IgnoreCase**)**;

// From the version group, get the version number

versionNumber = match.Groups**[**"version"**]**.Value;

**}**

**if** **(**versionNumber == "1"**)**

**{**

// if the version number is 1, then append V1 to the controller name.

// So at this point the, controller name will become EmployeesV1

controllerName = controllerName + "V1";

**}**

**else**

**{**

// if version number is 2, then append V2 to the controller name.

// So at this point the controller name will become EmployeesV2

controllerName = controllerName + "V2";

**}**

HttpControllerDescriptor controllerDescriptor;

**if** **(**controllers.TryGetValue**(**controllerName, out controllerDescriptor**))**

**{**

**return** controllerDescriptor;

**}**

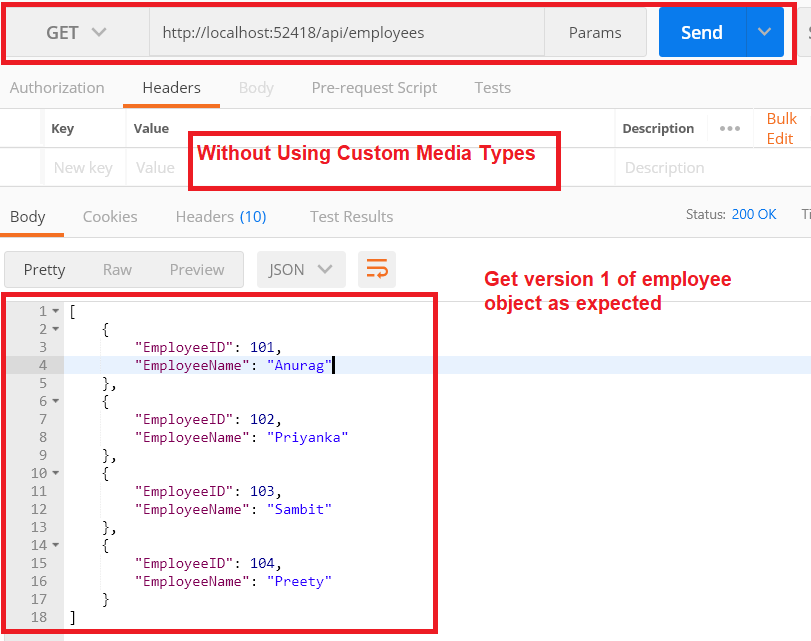
**return** **null**;

**}**

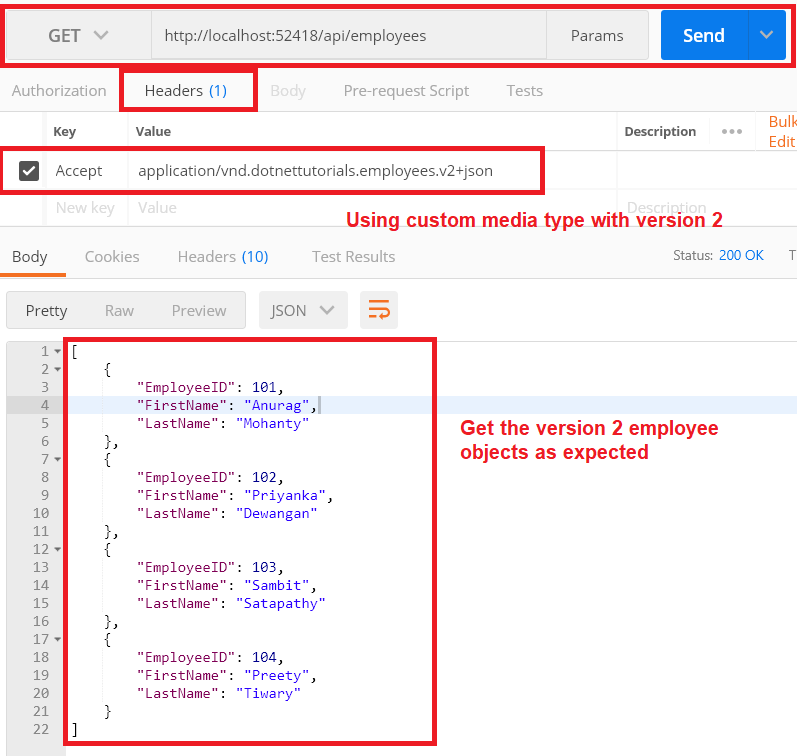
**}**

**}**

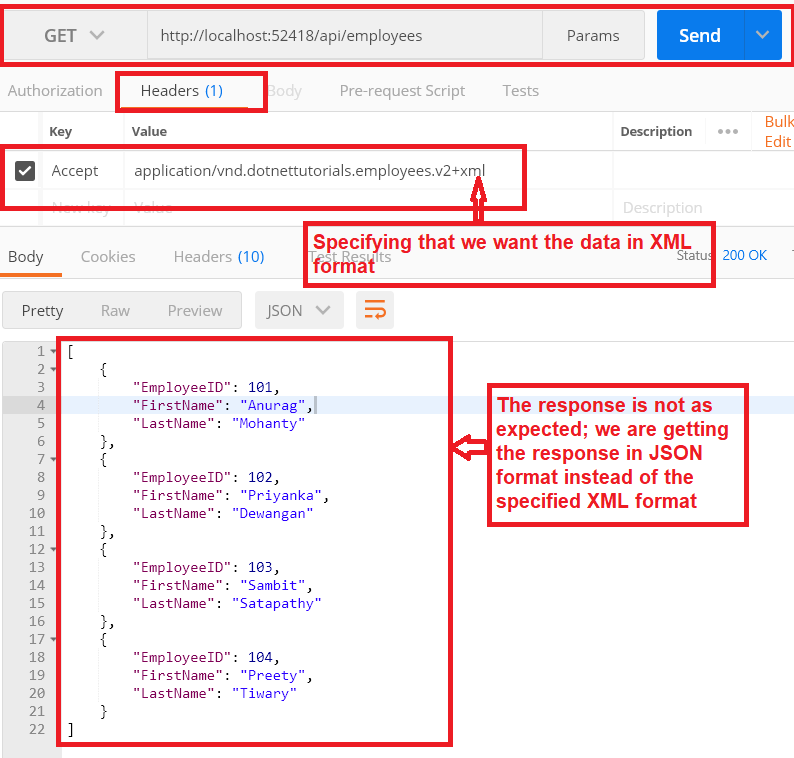
That’s it; we are done with our implementation. Build the solution and issue a request without using the custom media type from the postman. You will get version 1 of employee objects in JSON format as shown below.



Now specify the custom media type and set the version as 2 and issue a request, then you will get the version 2 of the employee objects as expected as shown in the below image.



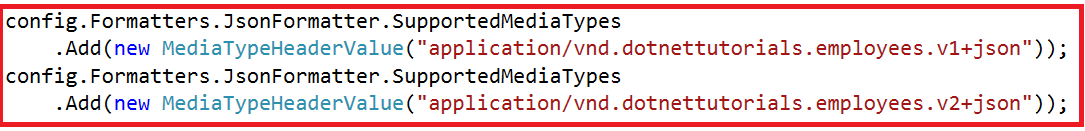
That’s work as expected. But there is a problem. The problem is if you specify that you want the data in XML format instead of JSON format in the request as shown below, then you will get the data in JSON format instead of the XML format.



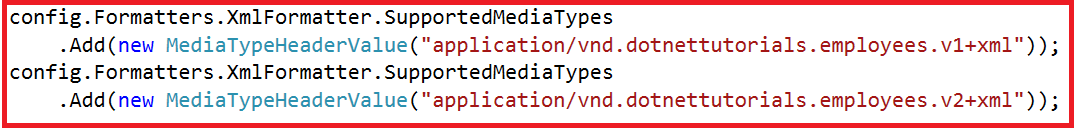
The reason is, we have not added our custom Media Type to the respective media type formatters i.e. **JsonFormatter** and **XmlFormatter**.

**Adding Custom Media Types:**

Include the following two lines of code within the **WebApiConfig.cs** file to add our custom media types to the **JsonFormatter**



Similarly, include the following two lines of code within the **WebApiConfig.cs** file to add the custom media types to the **XmlFormatter**



**So at this point, the complete code of the WebApiConfig.cs is given below.**

**using** *System.Net.Http.Headers;*

**using** *System.Web.Http;*

**using** *System.Web.Http.Dispatcher;*

**using** *WebAPIVersioning.Custom;*

**namespace** *WebAPIVersioning*

**{**

**public** **static** **class** WebApiConfig

**{**

**public** **static** **void** Register**(**HttpConfiguration config**)**

**{**

config.MapHttpAttributeRoutes**()**;

//Replacing default controller selector to our custom controller selector

config.Services.Replace**(**typeof**(**IHttpControllerSelector**)**,

new CustomControllerSelector**(**config**))**;

//Defineing the default route

config.Routes.MapHttpRoute**(**

name: "DefaultRoute",

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

defaults: new **{** id = RouteParameter.Optional **}**

**)**;

//Adding the custom media type to the JsonFormatter

config.Formatters.JsonFormatter.SupportedMediaTypes

.Add**(**new MediaTypeHeaderValue**(**"application/vnd.dotnettutorials.employees.v1+json"**))**;

config.Formatters.JsonFormatter.SupportedMediaTypes

.Add**(**new MediaTypeHeaderValue**(**"application/vnd.dotnettutorials.employees.v2+json"**))**;

//Adding the custom media type to the XmlFormatter

config.Formatters.XmlFormatter.SupportedMediaTypes

.Add**(**new MediaTypeHeaderValue**(**"application/vnd.dotnettutorials.employees.v1+xml"**))**;

config.Formatters.XmlFormatter.SupportedMediaTypes

.Add**(**new MediaTypeHeaderValue**(**"application/vnd.dotnettutorials.employees.v2+xml"**))**;

**}**

**}**

**}**

Now build the solution and issue a request by specifying that you want the data in XML format and you will get the data as expected as shown below.

