Practical Number 6

Web API Versioning Using URI

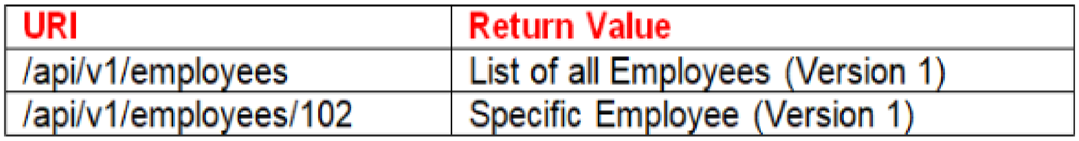
Why Web API versioning is required?

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?

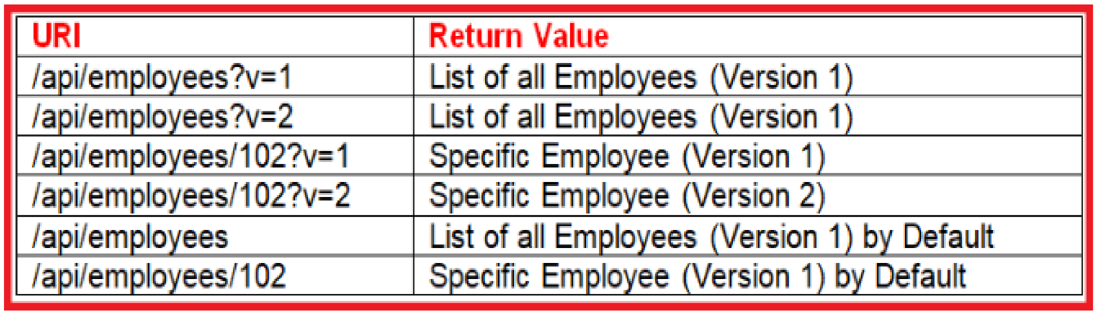
The different options that are available to maintain versioning are as follows

1. URI's



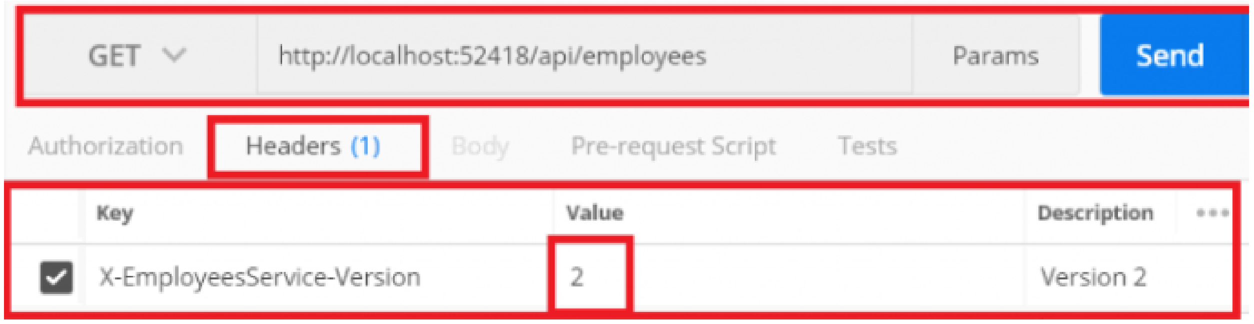
URI	Return Value
/api/v1/employees	List of all Employees (Version 1)
/api/v1/employees/102	Specific Employee (Version 1)

2. Query String



URI	Return Value
/api/employees?v=1	List of all Employees (Version 1)
/api/employees?v=2	List of all Employees (Version 1)
/api/employees/102?v=1	Specific Employee (Version 1)
/api/employees/102?v=2	Specific Employee (Version 2)
/api/employees	List of all Employees (Version 1) by Default
/api/employees/102	Specific Employee (Version 1) by Default

3. Version Header



GET http://localhost:52418/api/employees

	Key	Value	Description
✓	X-EmployeesService-Version	2	Version 2

4. Accept Header
5. Media Type

How to version a Web API using URI's?

I am going to discuss how to maintain versioning using URIs.

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.

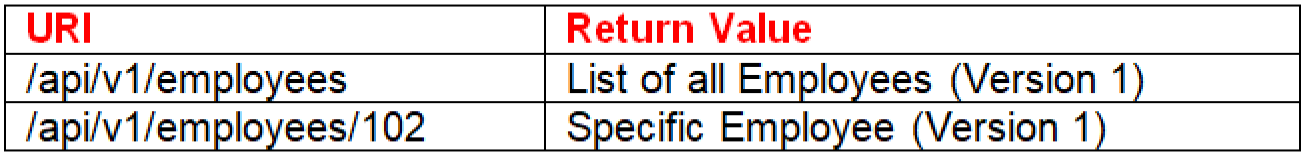
```
using System.Web.Http;

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.



URI	Return Value
/api/v1/employees	List of all Employees (Version 1)
/api/v1/employees/102	Specific Employee (Version 1)

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.

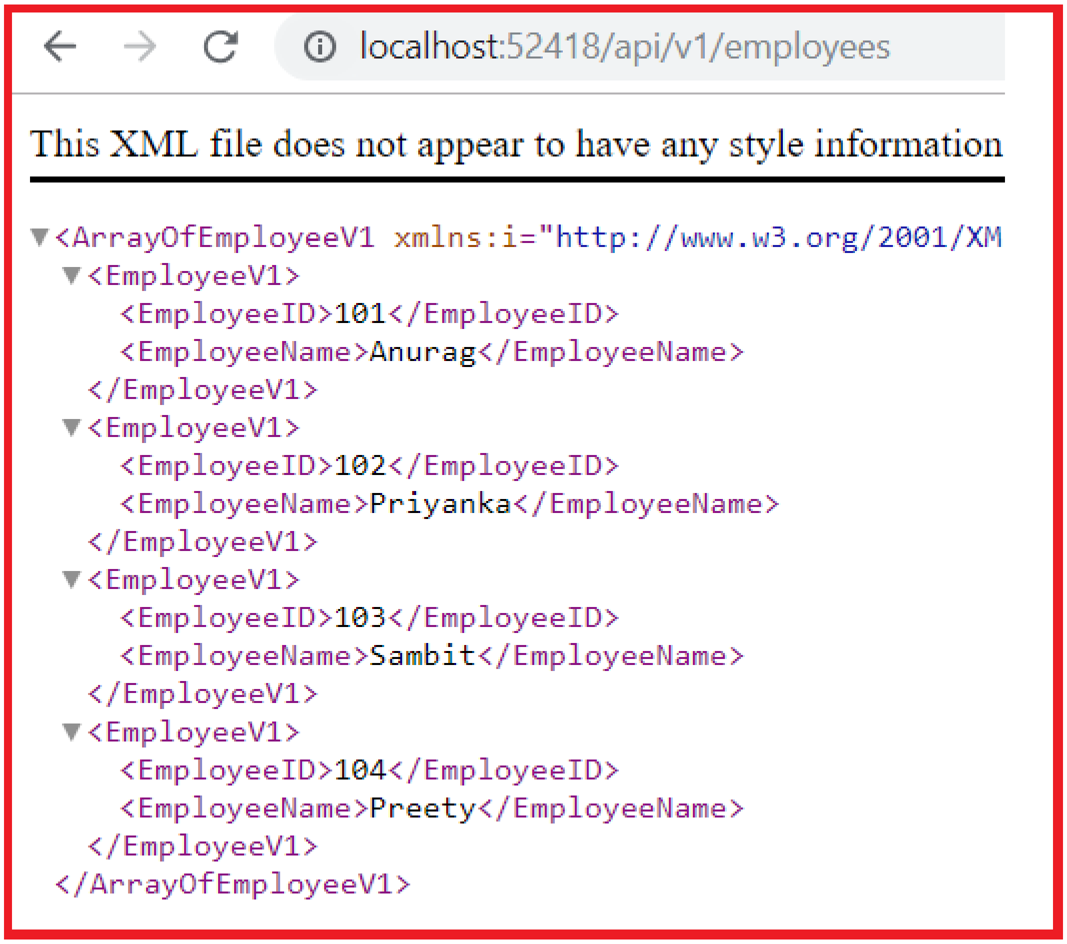
```
using System.Collections.Generic;

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" }

);

}

}

}
```

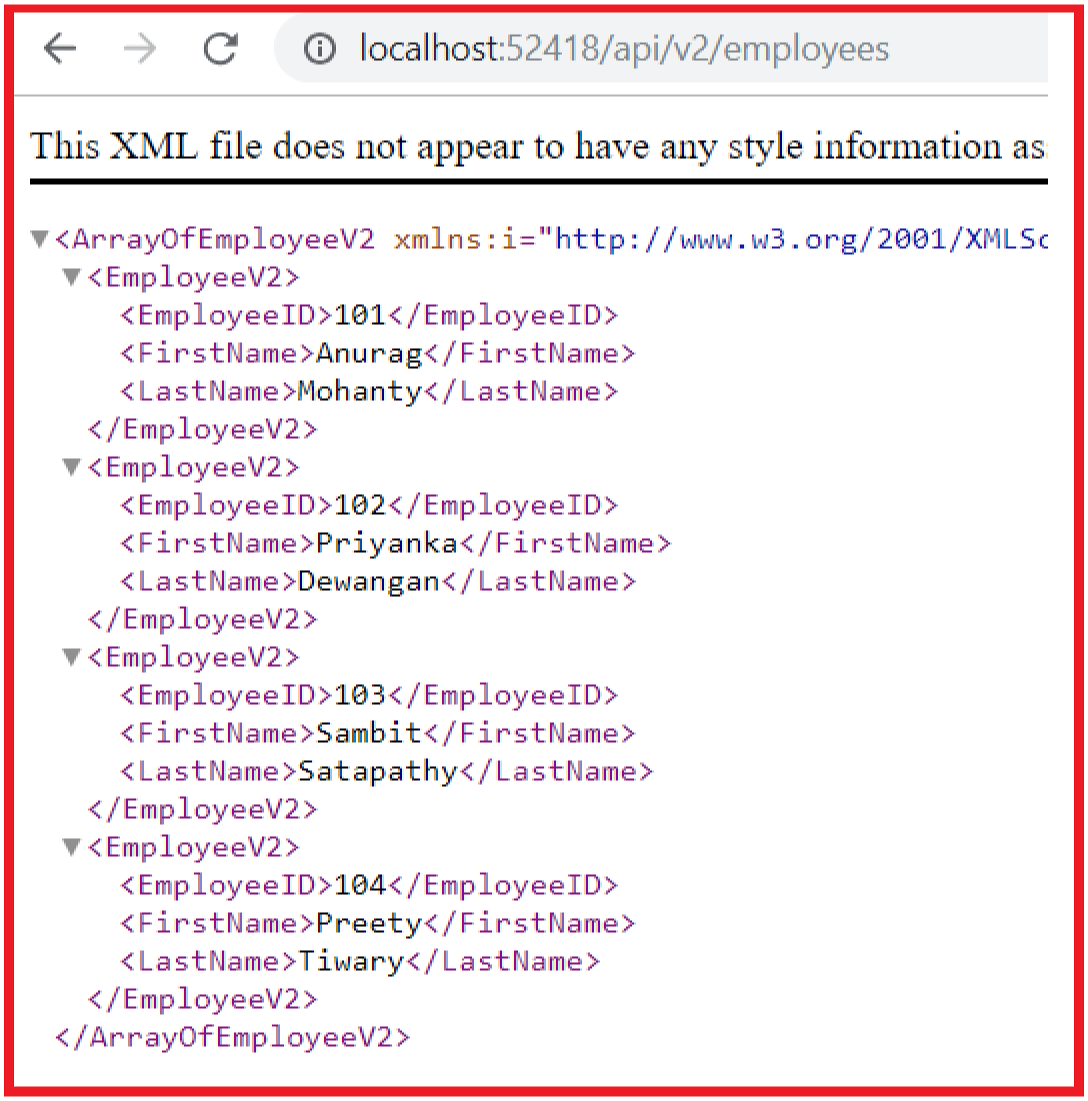
As shown in the above **WebApiConfig.cs** file, now we have 2 routes. Notice the route template, for each of the routes. The
Version 1 clients use **"/api/v1/employees/{id}"** route while the
Version 2 clients use **"/api/v2/employees/{id}"** route
That's it. We have implemented the versioning in our Web API service. So, if we navigate to **/api/v1/employees** – We get the employee with Employee Id and Name properties as shown below



This XML file does not appear to have any style information

```
▼<ArrayOfEmployeeV1 xmlns:i="http://www.w3.org/2001/XM
  ▼<EmployeeV1>
     <EmployeeID>101</EmployeeID>
     <EmployeeName>Anurag</EmployeeName>
   </EmployeeV1>
  ▼<EmployeeV1>
     <EmployeeID>102</EmployeeID>
     <EmployeeName>Priyanka</EmployeeName>
   </EmployeeV1>
  ▼<EmployeeV1>
     <EmployeeID>103</EmployeeID>
     <EmployeeName>Sambit</EmployeeName>
   </EmployeeV1>
  ▼<EmployeeV1>
     <EmployeeID>104</EmployeeID>
     <EmployeeName>Preety</EmployeeName>
   </EmployeeV1>
 </ArrayOfEmployeeV1>
```

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



This XML file does not appear to have any style information as

```
▼<ArrayOfEmployeeV2 xmlns:i="http://www.w3.org/2001/XMLSc
  ▼<EmployeeV2>
     <EmployeeID>101</EmployeeID>
     <FirstName>Anurag</FirstName>
     <LastName>Mohanty</LastName>
   </EmployeeV2>
  ▼<EmployeeV2>
     <EmployeeID>102</EmployeeID>
     <FirstName>Priyanka</FirstName>
     <LastName>Dewangan</LastName>
   </EmployeeV2>
  ▼<EmployeeV2>
     <EmployeeID>103</EmployeeID>
     <FirstName>Sambit</FirstName>
     <LastName>Satapathy</LastName>
   </EmployeeV2>
  ▼<EmployeeV2>
     <EmployeeID>104</EmployeeID>
     <FirstName>Preety</FirstName>
     <LastName>Tiwary</LastName>
   </EmployeeV2>
 </ArrayOfEmployeeV2>
```

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.