**Packages:**

1. ***Asp.Versioning.Mvc***
2. ***Asp.Versioning.Mvc.ApiExplorer***

***What is API Versioning?***

API Versioning is a technique we are implementing new API functionality without breaking the existing. It allows you to maintain backward compatibility while implanting new feature or changes to the API.

A diagram of a system

Description automatically generated

***When can we consider having a new API version?***

*o Changing the type of DTO property.*

*o Adding new validation on request.*

*o Renaming DTO properties.*

*o Removing DTO properties.*

*o Renaming end points.*

*o Removing end points.*

There are several ways we can Implement API versioning in core API.

1. **URL-Based Versioning**: In this approach, we can specify the version of API in URL. For Example, In URL we need to include the version number just like, *api/v1/Products*

2. **Header-Based Versioning**: In this technique, we must use a request header to specify the version number in header of the API.

3. **Query String Based Versioning**: As the name suggesting we can use query string to specify the version number.

4. **Media type versioning**: This option uses the media type to specify the version of the API. For example, “application/vnd.yourcompany.product-v1+json” in the Accept header.

Let’s go ahead and understand how we can implement the versioning in a Core API application.

*Prerequisites*:

1. Visual Studio 2022 (Community).
2. .NET 6 SDK

Let’s Jump into Visual Studio, Create an ASP.NET Core Web API project by choosing the template.



ASP.NET Core API Template

Once our project files are created, First and foremost, install the required packages i.e *Asp.Versioning.Mvc* and *Asp.Versioning.Mvc.ApiExplorer* (I always prefer to install the package version equal or higher to the .NET, In this case I have am using .NET 6 for this demonstration).

Go to Nuget package Manager. Frist install *Asp.Versioning.Mvc*

A screenshot of a computer

Description automatically generated

Next One *Asp.Versioning.Mvc.ApiExplorer*

A screenshot of a computer

Description automatically generated

Once our packages are installed, we must configure the versioning in program.cs file.

|  |
| --- |
| builder.Services.AddApiVersioning(option => {  option.AssumeDefaultVersionWhenUnspecified = true; //This ensures if client doesn't specify an API version. The default version should be considered.   option.DefaultApiVersion = new ApiVersion(1, 0); //This we set the default API version  option.ReportApiVersions = true; //The allow the API Version information to be reported in the client in the response header. This will be useful for the client to understand the version of the API they are interacting with.    //------------------------------------------------//  option.ApiVersionReader = ApiVersionReader.Combine(  new QueryStringApiVersionReader("api-version"),  new HeaderApiVersionReader("X-Version"),  new MediaTypeApiVersionReader("ver")); //This says how the API version should be read from the client's request, 3 options are enabled 1.Querystring, 2.Header, 3.MediaType.   //"api-version", "X-Version" and "ver" are parameter name to be set with version number in client before request the endpoints. }).AddApiExplorer(options => {  options.GroupNameFormat = "'v'VVV"; //The say our format of our version number “‘v’major[.minor][-status]”  options.SubstituteApiVersionInUrl = true; //This will help us to resolve the ambiguity when there is a routing conflict due to routing template one or more end points are same. }); |

Now let’s go and implement our end points. Here two important steps we need to follow, one is Api versions this Controller supports, We can define this with the help of ***[ApiVersion(“1.0”)],[ApiVersion(“2.0”)]*** and decorate at Controller level. Next, we need to define the method map to the Api Version at action method level with help of ***[MapToApiVersion(“1.0”)],[MapToApiVersion(“2.0”)]*** attributes respectively.

The **AddApiExplorer** method is helpful if you are using Swagger. It will fix the endpoint routes and substitute the API version route parameter.

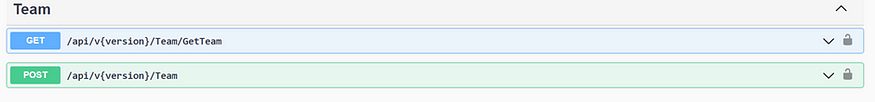
Now we will go the the Controller we want to add the version.

|  |
| --- |
| [ApiVersion("1.0")]  [ApiVersion("2.0")]  [Route("api/v{version:apiVersion}/[Controller]")] [ApiController] public class TeamController: ControllerBase {  [HttpGet("GetTeam")]  [MapToApiVersion("1.0")]  public IActionResult GetV1()  {  return Ok("V1 Get to be implemented");  }  [HttpGet("GetTeam")]  [MapToApiVersion("2.0")]  public IActionResult GetV2()  {  return Ok("V2 Get to be implemented");  }  [HttpPost]  public IActionResult Post(Team team)  {  return Ok("V1 Post to be implemented");  } } |

Here two versions have been mapped that is why we have ***[ApiVersion(“1.0”)] [ApiVersion(“2.0”)]*** and ***[MapToApiVersion(“1.0”)]*** and ***[MapToApiVersion(“2.0”)]*** this is helps to map the action method to their respective version. if tomorrow you wants to add one more version, we can add with ***[ApiVersion(“3.0”)]*** and Map it with help of ***[MapToApiVersion(“3.0”)]*** decorator.

Now Let’s run the application.

In swagger UI, we can see action method enabled with versioning. This is an mandatory we must provide the version number in order to access the our endpoints.



Passing version number in URL is only the first option way.

There are several other options we can pass the version number as well. However, at once we can utilize only one option.

[**Deprecating API Versions**](https://www.milanjovanovic.tech/blog/api-versioning-in-aspnetcore#deprecating-api-versions)

If you want to deprecate an old API version, you can set the **Deprecated** property on the **ApiVersion** attribute. The deprecated API versions will be reported using the **api-deprecated-versions** response header.

|  |
| --- |
| [ApiVersion(1, Deprecated = true)]  [ApiVersion(2)] |

A screenshot of a computer

Description automatically generated