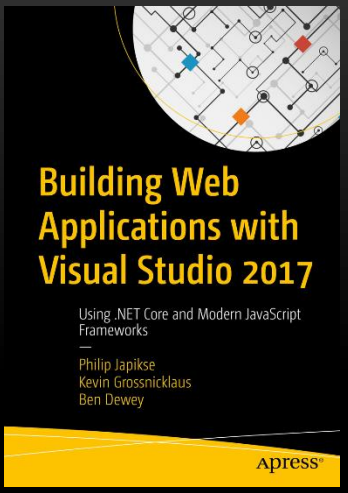
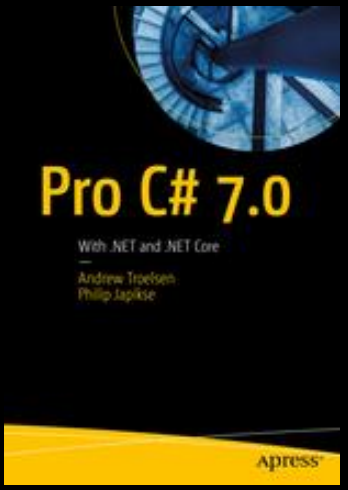
VERSION ASP.NET CORE APIS

Philip Japikse (@skimedic)
skimedic@outlook.com
www.skimedic.com/blog
Microsoft MVP, ASPInsider, MCSD, MCDBA, CSM, PSM, PSD
Consultant, Teacher, Writer

Phil.About()

- Consultant, Coach, Author, Teacher
  - Lynda.com (http://bit.ly/skimediclyndacourses)
  - Apress.com (http://bit.ly/apressbooks)
- Microsoft MVP, ASPInsider, MCSD, MCDBA, CSM, PSM, PSD
- Founder, Agile Conferences, Inc.
  - http://www.dayofagile.org
- President, Cincinnati .NET User's Group





DEFINING THE PROBLEM

DO YOU NEED TO VERSION?

- Your API is Public
- Your API needs updating
- Your API has more than one client OR
  - You plan on adding more
- You want to plan for the future (but not gold plate)

- Clients need to count on services being stable over time
- Business needs to add new features and make changes

MICROSOFT REST API GUIDELINES ON VERSIONING

VERSION FORMATS

- Services are versioned using the Major.Minor versioning scheme
  - 1.0, 2.0
- Services can opt for only the Major version - the ".0" is implied
  - V1 => v1.0
- Status (RC, Alpha, Beta, etc.) can be specified after the Minor version
  - 1.0-Alpha
- Grouping using YYYY-MM-DD
  - 2018-06-12.1.0-RC

https://github.com/Microsoft/api-guidelines/blob/vNext/Guidelines.md

VERSIONING OPTIONS

- Embed the version after the service root
  - https://www.skimedic.com/api/v1.0/classes
- Use a query string parameter
  - https://www.skimedic.com/api/classes?api-version=1.0
- Use Media Type/~~HTTP Headers~~
  - accept: text/plain;v=2.0

https://github.com/Microsoft/api-guidelines/blob/vNext/Guidelines.md

VERSIONING GUIDANCE

- Services located behind a DNS end point MUST use the same versioning mechanism
- Must present a consistent user experience
- Services must guarantee stability of the REST APIs
  - Naming and structure cannot change over time
- Major versions indicate previous version is deprecated
  - Documentation must indicate the support status of previous versions and provide a pth to the latest version

https://github.com/Microsoft/api-guidelines/blob/vNext/Guidelines.md

BREAKING CHANGES

- Versions must be incremented when API has breaking change
  - Optionally can update version on non-breaking changes
- Breaking changes include:
  - Removal or renaming APIs or API parameters
  - Changes in the behavior of an existing API
  - Changes in Error Codes and Fault Contracts
  - Anything that violates the Principle of Least Astonishment

PRINCIPLE OF LEAST ASTONISHMENT

- If a necessary feature has a high astonishment factor, it may be necessary to redesign the feature - 1984
- A component of a system should behave is a way that users expect it to behave
- For an API, function or method names intuitively match their behavior

https://en.wikipedia.org/wiki/Principle_of_least_astonishment

GROUP VERSIONING

- Group Versioning is an optional feature
  - Defined using the YYYY-MM-DD format
  - Uses the query string parameter mechanism
  - Does not replace the Major.Minor version format
- Allows for logical grouping of API endpoints under a common versioning moniker
  - Developers can lookup a single version and user it across related end points
  - API returns the greatest Major.Minor version in the group

ADDITIONAL CONSIDERATIONS

- Business logic
- Database schema
- Third-party services

VERSIONING ASP.NET CORE WEB SERVICES

GETTING STARTED

- Add Package
  - Microsoft.AspNetCore.Mvc.Versioning
- Add call to services.AddApiVersioning in ConfigureServices (Startup.cs)
- Configure API Versioning Options

API VERSIONING OPTIONS

DEFAULT VERSIONS

- AssumeDefaultVersionWhenUnspecified
  - Should only be used when adding versioning to an existing API
  - Returned version is configured with ApiVersionSelector
- DefaultApiVersion
  - Configured default value is 1.0
  - Can be set to another value

```
services.AddApiVersioning(o =>
{
   o.AssumeDefaultVersionWhenUnspecified = true;
   o.DefaultApiVersion = new ApiVersion(1,0);
});
```

APIVERSIONSELECTOR

- The **IApiVersionSelector** interface defines the behavior of how an API version is selected for a given request context.
- Default – the configured default
- Constant – always selects the specified version
- Current/Lowest Implementation – greatest/lowest version number

```
services.AddApiVersioning(o =>
{
   o.ApiVersionSelector = new DefaultApiVersionSelector(o);
   o.ApiVersionSelector = new ConstantApiVersionSelector(new ApiVersion(1,0));
   o.ApiVersionSelector = new CurrentImplementationApiVersionSelector( o );
   o.ApiVersionSelector = new LowestImplementedApiVersionSelector( o );
});
```

APIVERSIONREADER

- The **IApiVersionReader** interface defines the behavior of how an API version is read in its raw, unparsed form from the current HTTP request.
- The default API version reader is the **QueryStringApiVersionReader** class.
- QueryStringApiVersionReader reads the version from the query string.
  - The default query string parameter name is **api-version**.
- MediaTypeApiVersionReader reads the version from a HTTP media type request header.
  - The supported headers are **Content-Type** and **Accept**. If both headers are present, then **Content-Type** is preferred.
- Header ApiVersionReader – not compliant with the standards

APIVERSIONREADER

```
// svc?api-version=2.0
o.ApiVersionReader = new QueryStringApiVersionReader();
// svc?v=2.0
o.ApiVersionReader = new QueryStringApiVersionReader("v");

// Content-Type: application/json;v=2.0
o.ApiVersionReader = new MediaTypeApiVersionReader();
// Content-Type: application/json;version=2.0
o.ApiVersionReader = new MediaTypeApiVersionReader("version");
```

OPTIONS AVAILABLE

- ReportApiVersions
  - Enables sending api-supported-versions and api-deprecated-versions HTTP headers in responses.
  - This option is disabled by default
- Conventions
  - Allows you to construct API version conventions for each of your services as opposed to using .NET attributes.

VERSIONING

ADDING VERSIONING TO CONTROLLERS

- Use the ApiVersion attribute to add versioning

```
//Query String and Media Type
[ApiVersion( "2.0" )]
[Route( "api/helloworld" )]
public class HelloWorld2Controller : Controller
{
    …
}
//URL Versioning
[ApiVersion( "1.0" )]
[Route( "api/v{version:apiVersion}/[controller]" )]
public class HelloWorldController : Controller
{
    …
}
```

VERSION INTERLEAVING

- Use the ApiVersion attribute to add versioning

```
[ApiVersion( "2.0" )]
[ApiVersion( "3.0" )]
[Route( "api/v{version:apiVersion}/helloworld" )]
public class HelloWorld2Controller : Controller
{
    [HttpGet]
    public string Get() => "Hello world v2!";

    [HttpGet, MapToApiVersion( "3.0" )]
    public string GetV3() => "Hello world v3!";
}
```

DEPRECATING VERSIONS

- Add Deprecated to the ApiVersion attribute

```
[ApiVersion( "2.0" )]
[ApiVersion( "1.0", Deprecated = true )]
[Route( "api/[controller]" )]
public class HelloWorldController : Controller
{
    [HttpGet]
    public string Get() => "Hello world!"

    [HttpGet, MapToApiVersion( "2.0" )]
    public string GetV2() => "Hello world v2.0!";
}
```

GETTING THE REQUESTED VERSION INFORMATION

- User the GetRequestedApiVersion to return the requested version information

```
[ApiVersion("1.0")]
[ApiVersion("2.0")]
[Route("api/v{version:apiVersion}/[controller]/[action]")]
public class DifferentVersionsController : Controller
{
   [HttpGet]
   public string RequestedApiVersion() =>
      JsonConvert.SerializeObject(HttpContext.GetRequestedApiVersion());
}
```

VERSION DISCOVERY

- Create an HttpOptions method to expose the version information

```
// OPTIONS ~/api/myservice?api-version=[1.0|2.0|3.0]
[HttpOptions]
public IHttpActionResult Options()
{
    var response = new HttpResponseMessage( HttpStatusCode.OK );
    response.Content = new StringContent( string.Empty );
    response.Content.Headers.Add( "Allow", new[] { "GET", "POST",
"OPTIONS" } );
    response.Content.Headers.ContentType = null;
    return ResponseMessage( response );
}
```

VERSION NEUTRALITY

- Use the ApiVersionNeutral attribute to expose an endpoint to all versions

```
[ApiVersionNeutral]
[Route("api/v{version:apiVersion}/[controller]/[action]")]
public class HealthController : Controller
{
        [HttpGet]
        public string Ping() => "Ok";
}
```

VERSION DOCUMENTATION

API DOCUMENTATION

- The ASP.NET API versioning project provides several API explorer implementations to add versioning into your Swagger and Swashbuckle configurations.
- Add SwaggerGen to Configure Services (Startup.cs)
- Add Swagger and SwaggerUI to Configure (Startup.cs)
- To add Swashbuckle, must leverage Swashbuckle Extensibility model
  - Implement an IOperationFilter and add to Swagger

Contact Me

skimedic@outlook.com
www.skimedic.com/blog
www.twitter.com/skimedic

http://bit.ly/skimediclyndacourses
http://bit.ly/apressbooks

www.hallwayconversations.com

Questions?

Thank You!

Find the code at: https://github.com/skimedic/presentations