**XML + JSON Output for Web APIs in ASP .NET Core 3.1**

By Shahed C on June 22, 2020

[3 Replies](https://wakeupandcode.com/xml-json-output-for-web-apis-in-asp-net-core-3-1/#comments)



This is the twenty-fourth of a new [series of posts](https://wakeupandcode.com/aspnetcore/#aspnetcore2020) on ASP .NET Core 3.1 for 2020. In this series, we’ll cover 26 topics over a span of 26 weeks from January through June 2020, titled **ASP .NET Core A-Z!** To differentiate from the [2019 series](https://wakeupandcode.com/aspnetcore/#aspnetcore2019), the 2020 series will mostly focus on a growing single codebase ([NetLearner!](https://wakeupandcode.com/netlearner-on-asp-net-core-3-1/)) instead of new unrelated code snippets week.

Previous post:

* [Worker Service in .NET Core 3.1](https://wakeupandcode.com/worker-service-in-net-core-3-1/)

**NetLearner on GitHub**:

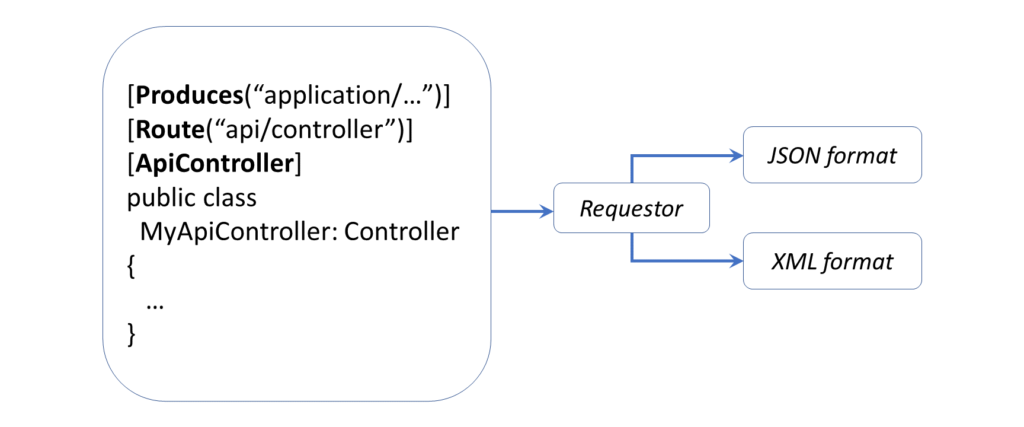
* Repository: <https://github.com/shahedc/NetLearnerApp>
* v0.24-alpha release: <https://github.com/shahedc/NetLearnerApp/releases/tag/v0.24-alpha>

**In this Article:**

* X is for XML + JSON Output
* Returning JsonResult and IActionResult
* Returning Complex Objects
* XML Output
* References

**X is for XML + JSON Output**

XML (eXtensible Markup Language) is a popular document format that has been used for a variety of applications over the years, including Microsoft Office documents, SOAP Web Services, application configuration and more. JSON (JavaScript Object Notation) was derived from object literals of JavaScript, but has also been used for storing data in both structured and unstructured formats, regardless of the language used. In fact, ASP .NET Core applications switched from XML-based .config files to JSON-based .json settings files for application configuration.

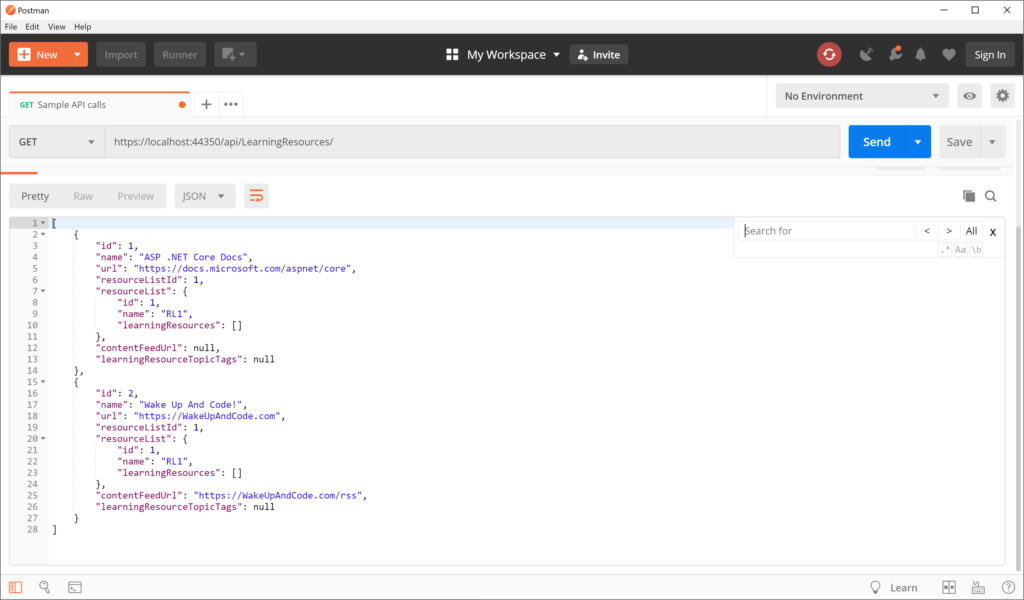
Returning XML/JSON format from a Web API

**Returning JsonResult and IActionResult**

Before we get into XML output for your Web API, let’s start off with JSON output first, and then we’ll get to XML. If you run the [Web API sample project](https://github.com/shahedc/NetLearnerApp/tree/main/src/NetLearner.Api) in the NetLearner repository, you’ll notice a [LearningResourcesController.cs](https://github.com/shahedc/NetLearnerApp/blob/main/src/NetLearner.Api/Controllers/LearningResourcesController.cs) file that represents a “Learning Resources Controller” that exposes API endpoints. These endpoints can serve up both JSON and XML results of Learning Resources, i.e. blog posts, tutorials, documentation, etc.

Run the application and navigate to the following endpoint in an API testing tool, e.g. [Postman](https://www.getpostman.com/):

* https://localhost:44350/api/LearningResources

Sample JSON data in Postman

This triggers a GET request by calling the [LearningResourcesController](https://github.com/shahedc/NetLearnerApp/blob/main/src/NetLearner.Api/Controllers/LearningResourcesController.cs)‘s **Get**() method:

  // GET: api/LearningResources  
 [HttpGet]  
 public JsonResult Get()  
 {  
 return new JsonResult(\_sampleRepository.LearningResources());  
 }

In this case, the **Json**() method returns a **JsonResult** object that serializes a list of Learning Resources. For simplicity, the \_sampleRepository object’s LearningResources() method (in SampleRepository.cs) returns a hard-coded list of **LearningResource**objects. Its implementation here isn’t important, because you would typically retrieve such values from a persistent data store, preferably through some sort of service class.

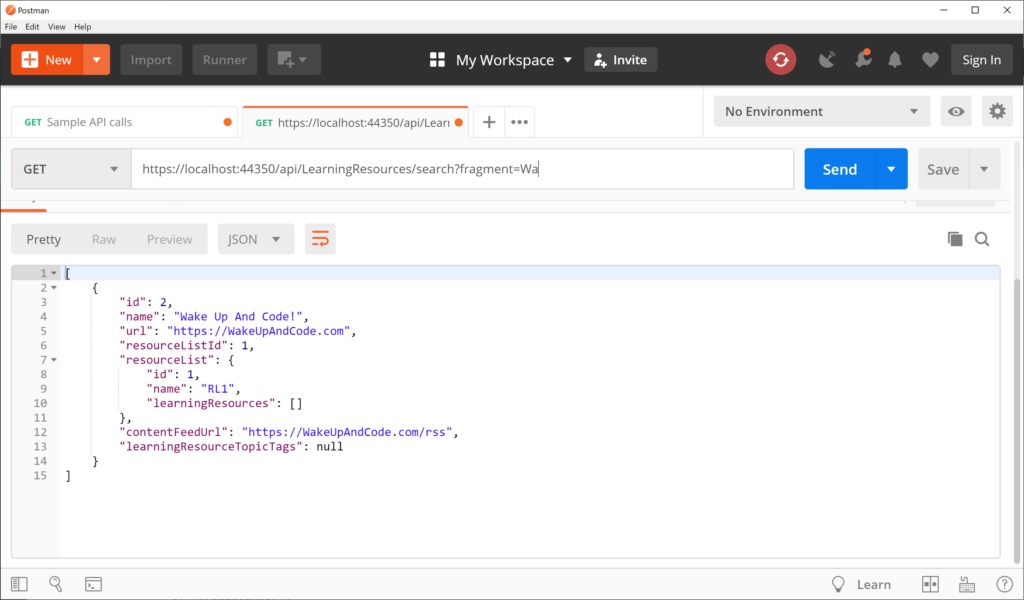
public List<LearningResource> LearningResources()  
{  
 ...   
 return new List<LearningResource>  
 {  
 new LearningResource  
 {  
 Id= 1,  
 Name= "ASP .NET Core Docs",  
 Url = "https://docs.microsoft.com/aspnet/core",  
 ...  
 },  
 ...   
 }  
}

The JSON result looks like the following, where a list of learning resources are returned:

[  
 {  
 "id": 1,  
 "name": "ASP .NET Core Docs",  
 "url": "https://docs.microsoft.com/aspnet/core",  
 "resourceListId": 1,  
 "resourceList": {  
 "id": 1,  
 "name": "RL1",  
 "learningResources": []  
 },  
 "contentFeedUrl": null,  
 "learningResourceTopicTags": null  
 },  
 {  
 "id": 2,  
 "name": "Wake Up And Code!",  
 "url": "https://WakeUpAndCode.com",  
 "resourceListId": 1,  
 "resourceList": {  
 "id": 1,  
 "name": "RL1",  
 "learningResources": []  
 },  
 "contentFeedUrl": "https://WakeUpAndCode.com/rss",  
 "learningResourceTopicTags": null  
 }  
]

Instead of specifically returning a **JsonResult**, you could also return a more generic **IActionResult**, which can still be interpreted as JSON. Run the application and navigate to the following endpoint, to include the action method “search” folllowed by a QueryString parameter “fragment” for a partial match.

* https://localhost:44350/api/LearningResources/**search**?**fragment**=Wa

 Sample JSON data with search string

This triggers a GET request by calling the [LearningResourceController](https://github.com/shahedc/NetLearnerApp/blob/main/src/NetLearner.Api/Controllers/LearningResourcesController.cs)‘s **Search**() method, with its fragment parameter set to “Wa” for a partial text search:

// GET: api/LearningResources/search?fragment=Wa  
[HttpGet("Search")]  
public IActionResult Search(string fragment)  
{  
 var result = \_sampleRepository.GetByPartialName(fragment);  
 if (!result.Any())  
 {  
 return NotFound(fragment);  
 }  
 return Ok(result);  
}

In this case, the **GetByPartialName**() method returns a **List** of **LearningResources** objects that are returned as JSON by default, with an HTTP 200 OK status. In case no results are found, the action method will return a 404 with the **NotFound**() method.

public List<LearningResource> GetByPartialName(string nameSubstring)  
{  
 return LearningResources()  
 .Where(lr => lr.Title  
 .IndexOf(nameSubstring, 0, StringComparison.CurrentCultureIgnoreCase) != -1)  
 .ToList();  
}

The JSON result looks like the following, which includes any learning resource that partially matches the string fragment provided:

[  
 {  
 "id": 2,  
 "name": "Wake Up And Code!",  
 "url": "https://WakeUpAndCode.com",  
 "resourceListId": 1,  
 "resourceList": {  
 "id": 1,  
 "name": "RL1",  
 "learningResources": []  
 },  
 "contentFeedUrl": "https://WakeUpAndCode.com/rss",  
 "learningResourceTopicTags": null  
 }  
]

**Returning Complex Objects**

An overloaded version of the **Get**() method takes in a “**listName**” string parameter to filter results by a list name for each learning resource in the repository. Instead of returning a **JsonResult** or **IActionResult**, this one returns a complex object (**LearningResource**) that contains properties that we’re interested in.

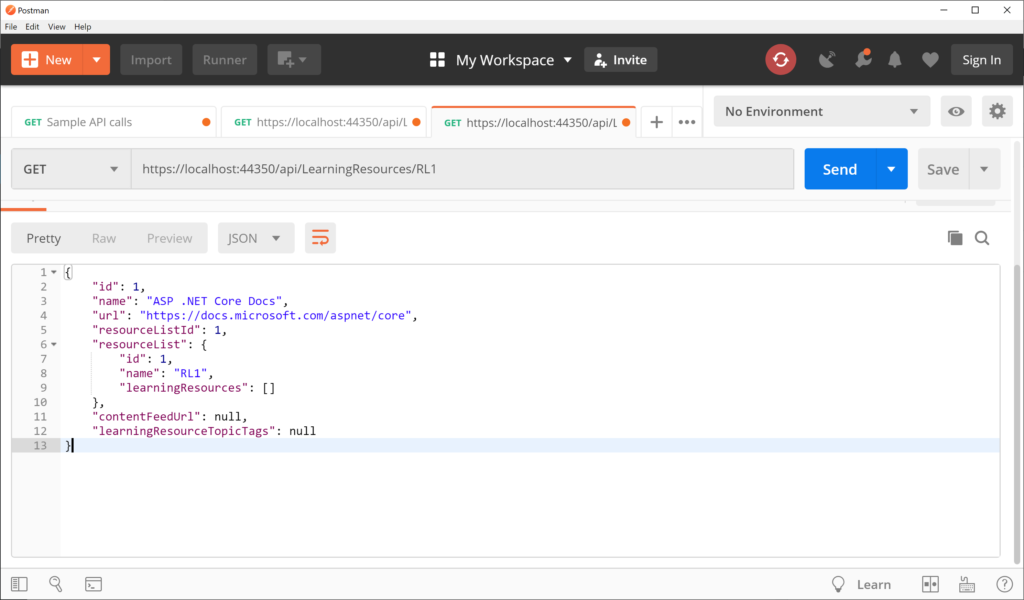
// GET api/LearningResources/RL1  
[HttpGet("{listName}")]  
public LearningResource Get(string listName)  
{  
 return \_sampleRepository.GetByListName(listName);  
}

The **GetByListName**() method in the [SampleRepository.cs](https://github.com/shahedc/NetLearnerApp/blob/main/src/NetLearner.Api/Infrastructure/SampleRepository.cs) class simply checks for a learning resource by the **listName** parameter and returns the first match. Again, the implementation is not particularly important, but it illustrates how you can pass in parameters to get back JSON results.

public LearningResource GetByListName(string listName)  
{  
 return LearningResources().FirstOrDefault(lr => lr.ResourceList.Name == listName);  
}

While the application is running, navigate to the following endpoint:

* https://localhost:44350/api/LearningResources/RL1

Sample JSON data with property filter

This triggers another GET request by calling the [LearningResourcesController](https://github.com/shahedc/NetLearnerApp/blob/main/src/NetLearner.Api/Controllers/LearningResourcesController.cs)‘s overloaded **Get**() method, with the listName parameter. When passing the list name “RL1”, this returns one item, as shown below:

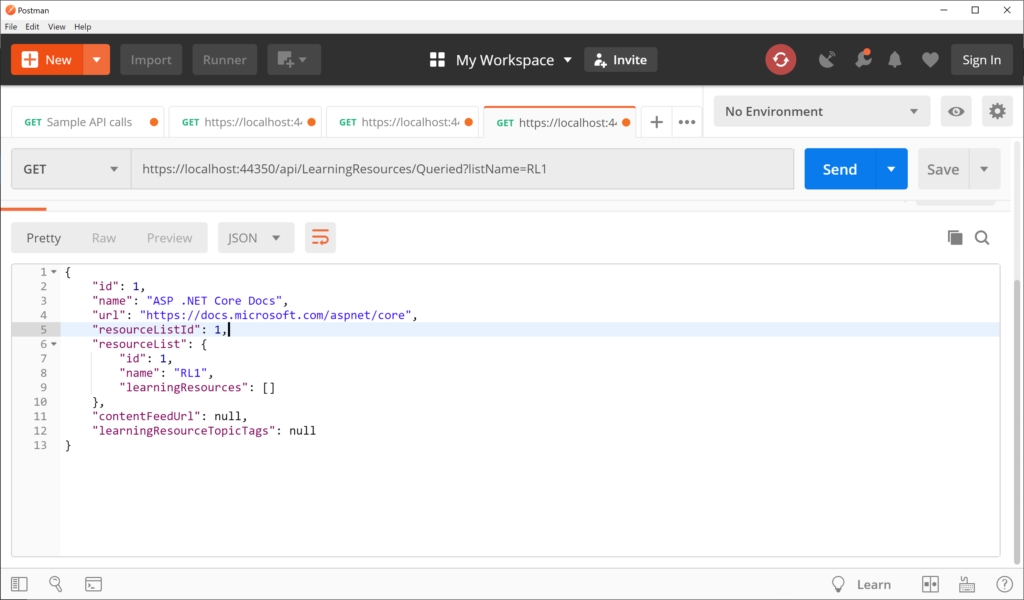
{  
 "id": 1,  
 "name": "ASP .NET Core Docs",  
 "url": "https://docs.microsoft.com/aspnet/core",  
 "resourceListId": 1,  
 "resourceList": {  
 "id": 1,  
 "name": "RL1",  
 "learningResources": []  
 },  
 "contentFeedUrl": null,  
 "learningResourceTopicTags": null  
}

Another example with a complex result takes in a similar parameter via QueryString and checks for an exact match with a specific property. In this case the **Queried**() action method calls the repository’s existing **GetByListName**() method to find a specific learning resource by its matching list name.

// GET: api/LearningResources/queried?listName=RL1  
[HttpGet("Queried")]  
public LearningResource Queried(string listName)  
{  
 return \_sampleRepository.GetByListName(listName);  
}

While the application is running, navigate to the following endpoint:

* https://localhost:44350/api/LearningResources/**Queried**?**listName**=RL1

Sample JSON data with QueryString parameter

This triggers a GET request by calling the [LearningResourcesController](https://github.com/shahedc/NetLearnerApp/blob/main/src/NetLearner.Api/Controllers/LearningResourcesController.cs)‘s **Queried**() method, with the **listName** parameter. When passing the list name “RL1”, this returns one item, as shown below:

{  
 "id": 1,  
 "name": "ASP .NET Core Docs",  
 "url": "https://docs.microsoft.com/aspnet/core",  
 "resourceListId": 1,  
 "resourceList": {  
 "id": 1,  
 "name": "RL1",  
 "learningResources": []  
 },  
 "contentFeedUrl": null,  
 "learningResourceTopicTags": null  
}

As you can see, the above result is in JSON format for the returned object.

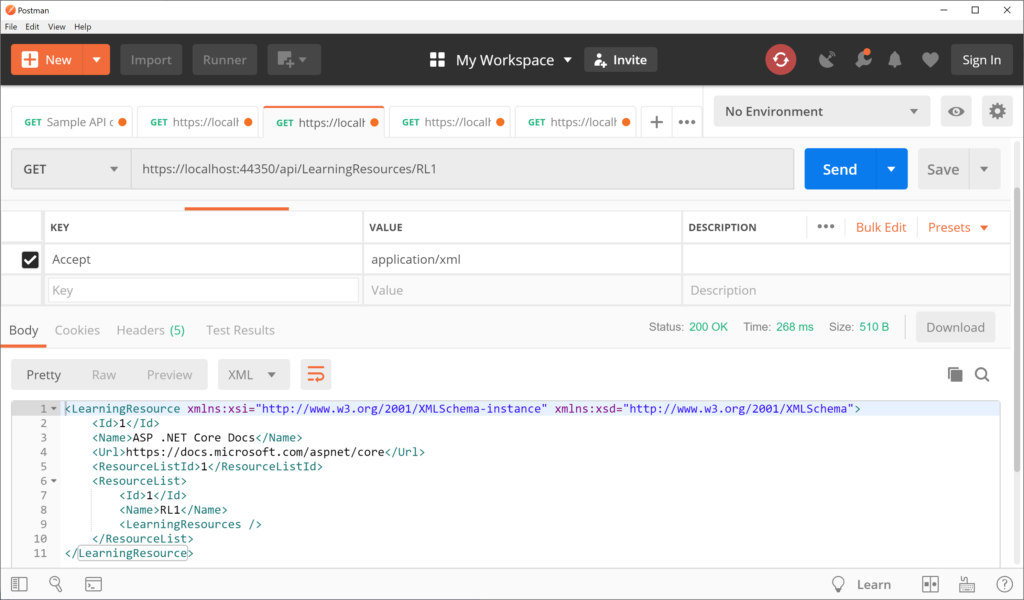
**XML Output**

*Wait a minute… with all these JSON results, when will we get to XML output?* Not to worry, there are multiple ways to get XML results while reusing the above code. First, update your Startup.cs file’s **ConfigureServices**() to include a call to services.AddControllers().**AddXmlSeralizerFormatters**():

public void ConfigureServices(IServiceCollection services)  
{  
 ...  
 services.AddControllers()  
 .AddXmlSerializerFormatters();  
 ...  
}

In Postman, set the request’s Accept header value to “application/xml” *before* requesting the endpoint, then run the application and navigate to the following endpoint once again:

* https://localhost:44350/api/LearningResources/RL1

XML-formatted results in Postman without code changes

This should provide the following XML results:

<LearningResource xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">  
 <Id>1</Id>  
 <Name>ASP .NET Core Docs</Name>  
 <Url>https://docs.microsoft.com/aspnet/core</Url>  
 <ResourceListId>1</ResourceListId>  
 <ResourceList>  
 <Id>1</Id>  
 <Name>RL1</Name>  
 <LearningResources />  
 </ResourceList>  
</LearningResource>

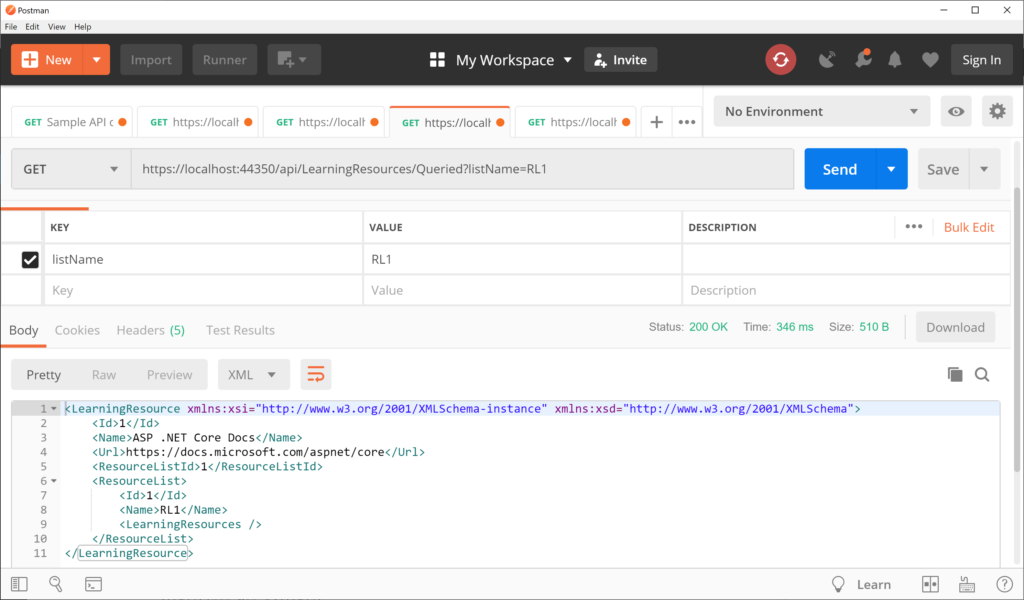
Since the action method returns a complex object, the result can easily be switched to XML simply by changing the Accept header value. In order to return XML using an IActionResult method, you should also use the [**Produces**] attribute, which can be set to “**application/xml**” at the API Controller level.

[Produces("application/xml")]  
[Route("api/[controller]")]  
[ApiController]  
public class LearningResourcesController : ControllerBase  
{  
 ...  
}

Then revisit the following endpoint, calling the search action method with the fragment parameter set to “ir”:

* https://localhost:44350/api/LearningResources/**Queried**?**listName**=RL1

At this point, it is no longer necessary to set the Accept header to “application/xml” (in Postman) during the request, since the [**Produces**] attribute is given priority over it.

XML-formatted output using Produces attribute

This should produces the following result , with a **LearningResource** object in XML:

<LearningResource xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">  
 <Id>1</Id>  
 <Name>ASP .NET Core Docs</Name>  
 <Url>https://docs.microsoft.com/aspnet/core</Url>  
 <ResourceListId>1</ResourceListId>  
 <ResourceList>  
 <Id>1</Id>  
 <Name>RL1</Name>  
 <LearningResources />  
 </ResourceList>  
</LearningResource>

As for the first **Get**() method returning **JsonResult**, you *can’t* override it with the [**Produces**] attribute or the **Accept** header value to change the result to XML format.

To recap, the order of precedence is as follows:

1. public **JsonResult** Get()
2. [**Produces**(“application/…”)]
3. **Accept**: “application/…”

**References**

* Format response data in ASP.NET Core Web API:  <https://docs.microsoft.com/en-us/aspnet/core/web-api/advanced/formatting>
* Postman Reference: <https://learning.postman.com/docs/postman/sending-api-requests/requests/>