Formatters Lab

Perform these labs on your own computer using Visual Studio 2022 to ensure you understand the lessons presented in the corresponding videos and lectures.

Lab 1: Formatting Attributes

Open the **Customer.cs** file and add the following using statement.

```
using System.Text.Json.Serialization;
```

Add the following attribute above the **MiddleName** property.

```
[JsonIgnore]
public string? MiddleName { get; set; }
```

Add the [JsonPropertyName] attribute to the following properties to rename the properties when they are serialized to JSON.

```
[JsonPropertyName("Company")]
public string? CompanyName { get; set; }
[JsonPropertyName("Email")]
public string? EmailAddress { get; set; }
```

Add the [JsonPropertyOrder] attribute to the **ModifiedDate** property to have it appear before all other properties in the serialized JSON.

```
[JsonPropertyOrder(-1)]
public DateTime ModifiedDate { get; set; }
```

Try it Out

Run the application and click on the **GET /api/Customer** button.

Notice the **ModifiedDate** property is listed first.

Notice the renaming of the two properties **CompanyName** and **EmailAddress**.

Also notice that the properties are not lower-case.

Notice the **MiddleName** property is no longer there.

You can now **Remove** the attributes and the **using** statement added in this demo.

Demo 2: Make Property Names PascalCase

Open the **Program.cs** file and locate the **AddControllers**() method.

Modify the call to this method.

```
// Configure ASP.NET to use the Controller model
var mvcBuilder = builder.Services.AddControllers();
```

Add a call to the AddJsonOptions()

```
// Configure JSON Options
mvcBuilder.AddJsonOptions(options =>
{
  // Make all property names start with upper-case
  options.JsonSerializerOptions.PropertyNamingPolicy =
null;
  // Ignore "readonly" fields
  options.JsonSerializerOptions.IgnoreReadOnlyProperties
= true;
});
```

Try it Out

Run the application and click on the **GET /api/Customer** button.

Notice the property names in the result set are now in **PascalCase**.

Demo 3: Ignore ReadOnly Properties

Open the **Customer.cs** file and add a new property named ***IsActive***

```
public bool IsActive { get; }
```

Run the application and see that the ***IsActive*** is added to the result set.

Open the **Program.cs** file and add another JSON option within the AddJsonOptions() method you added in the last lab.

```
// Ignore "readonly" fields
options.JsonSerializerOptions.IgnoreReadOnlyProperties =
true;
```

Try it Out

Run the application and click on the **GET /api/Customer** button.

Notice that **IsActive** property no longer appears in the result.

Demo 4: Returning XML

Open the **Program.cs** file and add a new call the AddXmlSerializerFormatters() after the AddJsonOptions() call. This allows you to return XML from your Web API calls.

```
// Configure XML Formatters
mvcBuilder.AddXmlSerializerFormatters();
```

Open the **CustomerController.cs** file and add the following attribute above the Get() method.

```
[Produces("application/xml")]
public ActionResult<IEnumerable<Customer>> Get()
```

Try it Out

Run the application and click on the **GET /api/Customer** button.

The result set should now be XML instead of JSON.

Demo 5: Consuming XML / Post Method

You can submit values to action methods as XML as well as return XML from your action methods. For example, in a POST method, you would normally pass a JSON object with the data to enter for the customer. However, you can also submit XML.

Open the **CustomerController.cs** file and add a new method named Post(). This method does not actually do anything with the data right now. Later in this course, you will actually submit and save a customer to the Customer table in a database.

```
[HttpPost]
[ProducesResponseType(StatusCodes.Status201Created)]
public ActionResult<Customer> Post(Customer entity)
{
  ActionResult<Customer> ret;

  // TODO: Insert Data into Data Store
  entity.ModifiedDate = DateTime.Now;

  ret = StatusCode(StatusCodes.Status201Created,
entity);

  return ret;
}
```

Though not shown, the default for input and output is the following:

```
[Consumes("application/json")]
[Produces("application/json")]
public ActionResult<Customer> Post([FromBody] Customer
entity)
```

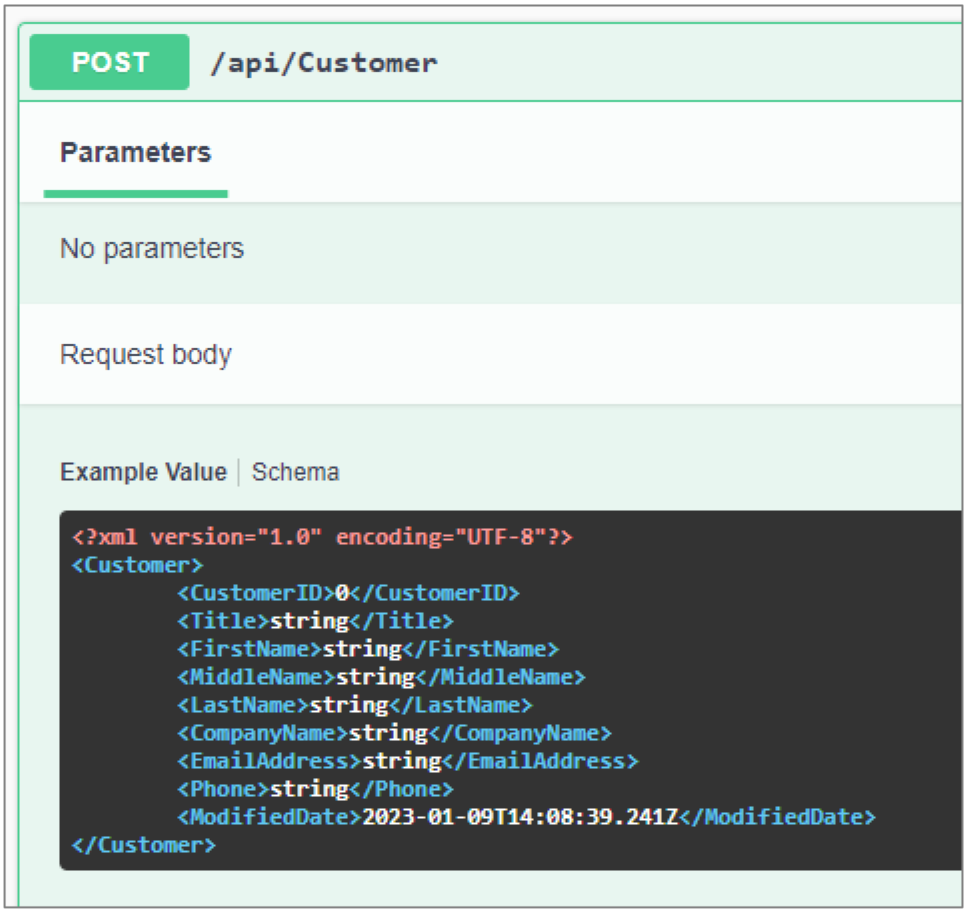
Add the following two attributes to the Post() method

```
[Consumes("application/xml")]
[Produces("application/xml")]
```

Try it Out

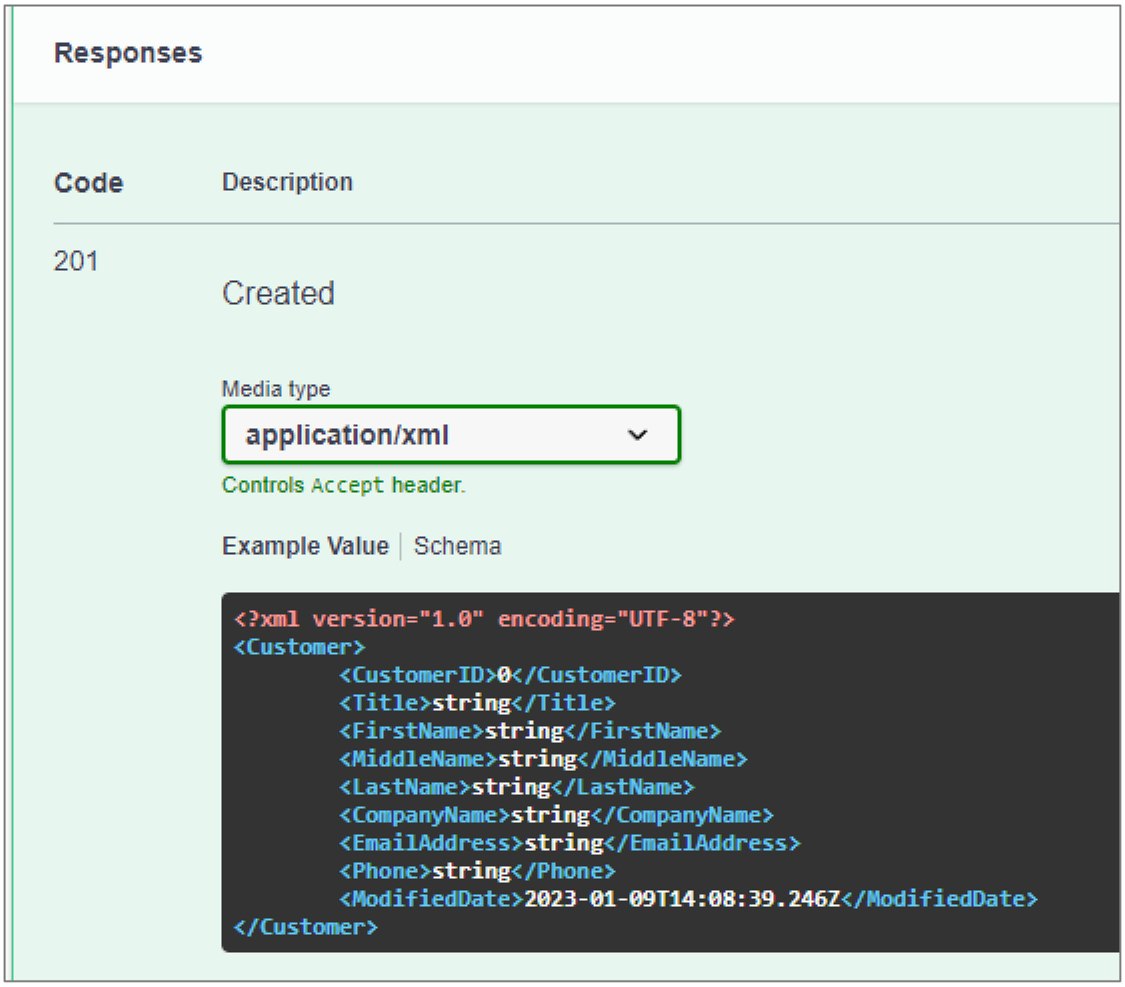
Run the application and click on the **POST /api/Customer** button.

Your browser should look something like the following:



```
<?xml version="1.0" encoding="UTF-8"?>
<Customer>
        <CustomerID>0</CustomerID>
        <Title>string</Title>
        <FirstName>string</FirstName>
        <MiddleName>string</MiddleName>
        <LastName>string</LastName>
        <CompanyName>string</CompanyName>
        <EmailAddress>string</EmailAddress>
        <Phone>string</Phone>
        <ModifiedDate>2023-01-09T14:08:39.241Z</ModifiedDate>
</Customer>
```

Also view the output



```
<?xml version="1.0" encoding="UTF-8"?>
<Customer>
	<CustomerID>0</CustomerID>
	<Title>string</Title>
	<FirstName>string</FirstName>
	<MiddleName>string</MiddleName>
	<LastName>string</LastName>
	<CompanyName>string</CompanyName>
	<EmailAddress>string</EmailAddress>
	<Phone>string</Phone>
	<ModifiedDate>2023-01-09T14:08:39.246Z</ModifiedDate>
</Customer>
```

Remove XML Serialization

Remove the **[Produces]** and **[Consumes]** attributes on both the Get() and Post() methods.