

Custom formatters in ASP.NET Core Web API

06/25/2020 • 5 minutes to read •      +2

In this article

[When to use custom formatters](#)

[Overview of how to use a custom formatter](#)

[How to create a custom formatter class](#)

[How to configure MVC to use a custom formatter](#)

[The complete VcardInputFormatter class](#)

[Test the app](#)

[Additional resources](#)

By [Kirk Larkin](#) and [Tom Dykstra](#) .

ASP.NET Core MVC supports data exchange in Web APIs using input and output formatters. Input formatters are used by [Model Binding](#). Output formatters are used to [format responses](#).

The framework provides built-in input and output formatters for JSON and XML. It provides a built-in output formatter for plain text, but doesn't provide an input formatter for plain text.

This article shows how to add support for additional formats by creating custom formatters. For an example of a custom plain text input formatter, see [TextPlainInputFormatter](#) on GitHub.

[View or download sample code](#) ([how to download](#))

When to use custom formatters

Use a custom formatter to add support for a content type that isn't handled by the built-in formatters.

Overview of how to use a custom formatter

To create a custom formatter:

- For serializing data sent to the client, create an output formatter class.

- For deserializing data received from the client, create an input formatter class.
- Add instances of formatter classes to the `InputFormatters` and `OutputFormatters` collections in [MvcOptions](#).

How to create a custom formatter class

To create a formatter:

- Derive the class from the appropriate base class. The sample app derives from [TextOutputFormatter](#) and [TextInputFormatter](#).
- Specify valid media types and encodings in the constructor.
- Override the [CanReadType](#) and [CanWriteType](#) methods.
- Override the [ReadRequestBodyAsync](#) and [WriteResponseBodyAsync](#) methods.

The following code shows the `VcardOutputFormatter` class from the [sample](#) :

C#

 Copy

```
public class VcardOutputFormatter : TextOutputFormatter
{
    public VcardOutputFormatter()
    {
        SupportedMediaTypes.Add(MediaTypeHeaderValue.Parse("text/vcard"));

        SupportedEncodings.Add(Encoding.UTF8);
        SupportedEncodings.Add(Encoding.Unicode);
    }

    protected override bool CanWriteType(Type type)
    {
        return typeof(Contact).IsAssignableFrom(type) ||
            typeof(IEnumerable<Contact>).IsAssignableFrom(type);
    }

    public override async Task WriteResponseBodyAsync(
        OutputFormatterWriteContext context, Encoding selectedEncoding)
    {
        var httpContext = context.HttpContext;
        var serviceProvider = httpContext.RequestServices;

        var logger =
            serviceProvider.GetRequiredService<ILogger<VcardOutputFormatter>>();
        var buffer = new StringBuilder();

        if (context.Object is IEnumerable<Contact> contacts)
        {
```

```
        foreach (var contact in contacts)
        {
            FormatVcard(buffer, contact, logger);
        }
    }
    else
    {
        FormatVcard(buffer, (Contact)context.Object, logger);
    }

    await httpContext.Response.WriteAsync(buffer.ToString());
}

private static void FormatVcard(
    StringBuilder buffer, Contact contact, ILogger logger)
{
    buffer.AppendLine("BEGIN:VCARD");
    buffer.AppendLine("VERSION:2.1");
    buffer.AppendLine($"N:{contact.LastName};{contact.FirstName}");
    buffer.AppendLine($"FN:{contact.FirstName} {contact.LastName}");
    buffer.AppendLine($"UID:{contact.Id}");
    buffer.AppendLine("END:VCARD");

    logger.LogInformation("Writing {FirstName} {LastName}",
        contact.FirstName, contact.LastName);
}
}
```

Derive from the appropriate base class

For text media types (for example, vCard), derive from the [TextInputFormatter](#) or [TextOutputFormatter](#) base class.

C#

 Copy

```
public class VcardOutputFormatter : TextOutputFormatter
```

For binary types, derive from the [InputFormatter](#) or [OutputFormatter](#) base class.

Specify valid media types and encodings

In the constructor, specify valid media types and encodings by adding to the `SupportedMediaTypes` and `SupportedEncodings` collections.

C#

 Copy

```
public VcardOutputFormatter()  
{  
    SupportedMediaTypes.Add(MediaTypeHeaderValue.Parse("text/vcard"));  
  
    SupportedEncodings.Add(Encoding.UTF8);  
    SupportedEncodings.Add(Encoding.Unicode);  
}
```

A formatter class can **not** use constructor injection for its dependencies. For example, `ILogger<VcardOutputFormatter>` cannot be added as a parameter to the constructor. To access services, use the context object that gets passed in to the methods. A code example in this article and the [sample](#) show how to do this.

Override CanReadType and CanWriteType

Specify the type to deserialize into or serialize from by overriding the `CanReadType` or `CanWriteType` methods. For example, creating vCard text from a `Contact` type and vice versa.

C#

 Copy

```
protected override bool CanWriteType(Type type)  
{  
    return typeof(Contact).IsAssignableFrom(type) ||  
           typeof(IEnumerable<Contact>).IsAssignableFrom(type);  
}
```

The CanWriteResult method

In some scenarios, `CanWriteResult` must be overridden rather than `CanWriteType`. Use `CanWriteResult` if the following conditions are true:

- The action method returns a model class.
- There are derived classes which might be returned at runtime.
- The derived class returned by the action must be known at runtime.

For example, suppose the action method:

- Signature returns a `Person` type.
- Can return a `Student` or `Instructor` type that derives from `Person`.

For the formatter to handle only `Student` objects, check the type of `Object` in the context object provided to the `CanWriteResult` method. When the action method returns `ActionResult`:

- It's not necessary to use `CanWriteResult`.
- The `CanWriteType` method receives the runtime type.

Override `ReadRequestBodyAsync` and `WriteResponseBodyAsync`

Deserialization or serialization is performed in `ReadRequestBodyAsync` or `WriteResponseBodyAsync`. The following example shows how to get services from the dependency injection container. Services can't be obtained from constructor parameters.

C#

 Copy

```
public override async Task WriteResponseBodyAsync(
    OutputFormatterWriteContext context, Encoding selectedEncoding)
{
    var httpContext = context.HttpContext;
    var serviceProvider = httpContext.RequestServices;

    var logger =
        serviceProvider.GetRequiredService<ILogger<VcardOutputFormatter>>();
    var buffer = new StringBuilder();

    if (context.Object is IEnumerable<Contact> contacts)
    {
        foreach (var contact in contacts)
        {
            FormatVcard(buffer, contact, logger);
        }
    }
    else
    {
        FormatVcard(buffer, (Contact)context.Object, logger);
    }

    await httpContext.Response.WriteAsync(buffer.ToString());
}

private static void FormatVcard(
    StringBuilder buffer, Contact contact, ILogger logger)
{
    buffer.AppendLine("BEGIN:VCARD");
    buffer.AppendLine("VERSION:2.1");
```

```
buffer.AppendLine($"N:{contact.LastName};{contact.FirstName}");
buffer.AppendLine($"FN:{contact.FirstName} {contact.LastName}");
buffer.AppendLine($"UID:{contact.Id}");
buffer.AppendLine("END:VCARD");

logger.LogInformation("Writing {FirstName} {LastName}",
    contact.FirstName, contact.LastName);
}
```

How to configure MVC to use a custom formatter

To use a custom formatter, add an instance of the formatter class to the `InputFormatters` or `OutputFormatters` collection.

C#

 Copy

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllers(options =>
    {
        options.InputFormatters.Insert(0, new VcardInputFormatter());
        options.OutputFormatters.Insert(0, new VcardOutputFormatter());
    });
}
```

Formatters are evaluated in the order you insert them. The first one takes precedence.

The complete `VcardInputFormatter` class

The following code shows the `VcardInputFormatter` class from the [sample](#) :

C#

 Copy

```
public class VcardInputFormatter : TextInputFormatter
{
    public VcardInputFormatter()
    {
        SupportedMediaTypes.Add(MediaTypeHeaderValue.Parse("text/vcard"));

        SupportedEncodings.Add(Encoding.UTF8);
        SupportedEncodings.Add(Encoding.Unicode);
    }

    protected override bool CanReadType(Type type)
```

```
{
    return type == typeof(Contact);
}

public override async Task<InputFormatterResult> ReadRequestBodyAsync(
    InputFormatterContext context, Encoding effectiveEncoding)
{
    var httpContext = context.HttpContext;
    var serviceProvider = httpContext.RequestServices;

    var logger =
serviceProvider.GetRequiredService<ILogger<VcardInputFormatter>>();

    using var reader = new StreamReader(httpContext.Request.Body,
effectiveEncoding);
    string nameLine = null;

    try
    {
        await ReadLineAsync("BEGIN:VCARD", reader, context, logger);
        await ReadLineAsync("VERSION:", reader, context, logger);

        nameLine = await ReadLineAsync("N:", reader, context, logger);

        var split = nameLine.Split(";".ToCharArray());
        var contact = new Contact
        {
            LastName = split[0].Substring(2),
            FirstName = split[1]
        };

        await ReadLineAsync("FN:", reader, context, logger);
        await ReadLineAsync("END:VCARD", reader, context, logger);

        logger.LogInformation("nameLine = {nameLine}", nameLine);

        return await InputFormatterResult.SuccessAsync(contact);
    }
    catch
    {
        logger.LogError("Read failed: nameLine = {nameLine}", nameLine);
        return await InputFormatterResult.FailureAsync();
    }
}

private static async Task<string> ReadLineAsync(
    string expectedText, StreamReader reader, InputFormatterContext
context,
    ILogger logger)
{
    var line = await reader.ReadLineAsync();
```

```
        if (!line.StartsWith(expectedText))
        {
            var errorMessage = $"Looked for '{expectedText}' and got '{line}'";


            context.ModelState.TryAddModelError(context.ModelName,
            errorMessage);
            logger.LogError(errorMessage);

            throw new Exception(errorMessage);
        }

        return line;
    }
}
```

Test the app

Run the [sample app for this article](#) , which implements basic vCard input and output formatters. The app reads and writes vCards similar to the following:

	 Copy
BEGIN:VCARD VERSION:2.1 N:Davolio;Nancy FN:Nancy Davolio END:VCARD	

To see vCard output, run the app and send a Get request with Accept header `text/vcard` to `https://localhost:5001/api/contacts`.

To add a vCard to the in-memory collection of contacts:

- Send a Post request to `/api/contacts` with a tool like Postman.
- Set the Content-Type header to `text/vcard`.
- Set vCard text in the body, formatted like the preceding example.

Additional resources

- [Format response data in ASP.NET Core Web API](#)
- [Manage Protobuf references with dotnet-grpc](#)

Is this page helpful?

 Yes  No
