

Media Formatters in ASP.NET Web API 2

📅 01/20/2014 ⌚ 4 minutes to read Contributors     

In this article

[Internet Media Types](#)

[Example: Creating a CSV Media Formatter](#)

[Adding a Media Formatter to the Web API Pipeline](#)

[Character Encodings](#)

by [Mike Wasson](#)

This tutorial shows how to support additional media formats in ASP.NET Web API.


Internet Media Types

A media type, also called a MIME type, identifies the format of a piece of data. In HTTP, media types describe the format of the message body. A media type consists of two strings, a type and a subtype. For example:

- text/html
- image/png
- application/json

When an HTTP message contains an entity-body, the Content-Type header specifies the format of the message body. This tells the receiver how to parse the contents of the message body.

For example, if an HTTP response contains a PNG image, the response might have the following headers.

console	 Copy
HTTP/1.1 200 OK Content-Length: 95267 Content-Type: image/png	

When the client sends a request message, it can include an Accept header. The Accept header tells the server which media type(s) the client wants from the server. For example:

console	 Copy
---------	--

```
Accept: text/html,application/xhtml+xml,application/xml
```

This header tells the server that the client wants either HTML, XHTML, or XML.

The media type determines how Web API serializes and deserializes the HTTP message body. Web API has built-in support for XML, JSON, BSON, and form-urlencoded data, and you can support additional media types by writing a *media formatter*.

To create a media formatter, derive from one of these classes:

- [MediaTypeFormatter](#). This class uses asynchronous read and write methods.
- [BufferedMediaTypeFormatter](#). This class derives from **MediaTypeFormatter** but uses synchronous read/write methods.

Deriving from **BufferedMediaTypeFormatter** is simpler, because there is no asynchronous code, but it also means the calling thread can block during I/O.

Example: Creating a CSV Media Formatter

The following example shows a media type formatter that can serialize a Product object to a comma-separated values (CSV) format. This example uses the Product type defined in the tutorial [Creating a Web API that Supports CRUD Operations](#). Here is the definition of the Product object:

C#

 Copy

```
namespace ProductStore.Models
{
    public class Product
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Category { get; set; }
        public decimal Price { get; set; }
    }
}
```

To implement a CSV formatter, define a class that derives from **BufferedMediaTypeFormatter**:

C#

 Copy

```
namespace ProductStore.Formatters
using System;
using System.Collections.Generic;
```

```
using System.IO;
using System.Net.Http;
using System.Net.Http.Formatting;
using System.Net.Http.Headers;
using ProductStore.Models;

namespace ProductStore.Formatters
{
    public class ProductCsvFormatter : BufferedMediaTypeFormatter
    {
    }
}
```

In the constructor, add the media types that the formatter supports. In this example, the formatter supports a single media type, "text/csv":

C#

 Copy

```
public ProductCsvFormatter()
{
    // Add the supported media type.
    SupportedMediaTypes.Add(new MediaTypeHeaderValue("text/csv"));
}
```

Override the **CanWriteType** method to indicate which types the formatter can serialize:

C#

 Copy

```
public override bool CanWriteType(System.Type type)
{
    if (type == typeof(Product))
    {
        return true;
    }
    else
    {
        Type enumerableType = typeof(IEnumerable<Product>);
        return enumerableType.IsAssignableFrom(type);
    }
}
```

In this example, the formatter can serialize single `Product` objects as well as collections of `Product` objects.

Similarly, override the **CanReadType** method to indicate which types the formatter can deserialize. In this example, the formatter does not support deserialization, so the method simply returns **false**.

C#

 Copy

```
public override bool CanReadType(Type type)
{
    return false;
}
```

Finally, override the **WriteToStream** method. This method serializes a type by writing it to a stream. If your formatter supports deserialization, also override the **ReadFromStream** method.

C#

 Copy

```
public override void WriteToStream(Type type, object value, Stream writeStream, HttpContent
{
    using (var writer = new StreamWriter(writeStream))
    {
        var products = value as IEnumerable<Product>;
        if (products != null)
        {
            foreach (var product in products)
            {
                WriteItem(product, writer);
            }
        }
        else
        {
            var singleProduct = value as Product;
            if (singleProduct == null)
            {
                throw new InvalidOperationException("Cannot serialize type");
            }
            WriteItem(singleProduct, writer);
        }
    }
}

// Helper methods for serializing Products to CSV format.
private void WriteItem(Product product, StreamWriter writer)
{
    writer.WriteLine("{0},{1},{2},{3}", Escape(product.Id),
        Escape(product.Name), Escape(product.Category), Escape(product.Price));
}

static char[] _specialChars = new char[] { ',', '\n', '\r', '"' };

private string Escape(object o)
{
    if (o == null)
    {

```

```
        return "";  
    }  
    string field = o.ToString();  
    if (field.IndexOfAny(_specialChars) != -1)  
    {  
        // Delimit the entire field with quotes and replace embedded quotes with "".  
        return String.Format("\"{0}\"", field.Replace("\"", "\"\""));  
    }  
    else return field;  
}
```

Adding a Media Formatter to the Web API Pipeline

To add a media type formatter to the Web API pipeline, use the **Formatters** property on the **HttpConfiguration** object.

C#

 Copy

```
public static void ConfigureApis(HttpConfiguration config)  
{  
    config.Formatters.Add(new ProductCsvFormatter());  
}
```

Character Encodings

Optionally, a media formatter can support multiple character encodings, such as UTF-8 or ISO 8859-1.

In the constructor, add one or more [System.Text.Encoding](https://docs.microsoft.com/en-us/dotnet/api/system.text.encoding) types to the **SupportedEncodings** collection. Put the default encoding first.

C#

 Copy

```
public ProductCsvFormatter()  
{  
    SupportedMediaTypes.Add(new MediaTypeHeaderValue("text/csv"));  
  
    // New code:  
    SupportedEncodings.Add(new UTF8Encoding(encoderShouldEmitUTF8Identifier: false));  
    SupportedEncodings.Add(Encoding.GetEncoding("iso-8859-1"));  
}
```

In the **WriteToStream** and **ReadFromStream** methods, call [MediaTypeFormatter.SelectCharacterEncoding](#) to select the preferred character encoding. This method matches the request headers against the list of supported encodings. Use the returned **Encoding** when you read or write from the stream:

C# Copy

```
public override void WriteToStream(Type type, object value, Stream writeStream, HttpContent
{
    Encoding effectiveEncoding = SelectCharacterEncoding(content.Headers);

    using (var writer = new StreamWriter(writeStream, effectiveEncoding))
    {
        // Write the object (code not shown)
    }
}
```

Note

The feedback system for this content will be changing soon. Old comments will not be carried over. If content within a comment thread is important to you, please save a copy. For more information on the upcoming change, [we invite you to read our blog post](#).