



MICROSOFT ARCHITECT

By Joydip Kanjilal, Columnist, InfoWorld
APR 20, 2020 3:00 AM PDT

How to export data to Excel in ASP.NET Core 3.0

Learn how to use the ClosedXML NuGet package to export data as a CSV or XLSX file for Excel from an ASP.NET Core application

When building web applications, you will often need to import or export data from or to Word or Excel documents. There are several ways to achieve this, and plenty of NuGet packages to work with Word or Excel. This article discusses how we can work with ClosedXML in ASP.NET Core to export data to Excel.

To work with the code examples provided in this article, you should have Visual Studio 2019 installed in your system. If you don't already have a copy, you can download Visual Studio 2019 [here](#).

Create an ASP.NET Core MVC project in Visual Studio

First off, let's create an ASP.NET Core project in Visual Studio 2019. Assuming Visual Studio 2019 is installed in your system, follow the steps outlined below to create a new ASP.NET Core project in Visual Studio.

[[Also on InfoWorld: How to build gRPC applications in ASP.NET Core](#)]

1. Launch the Visual Studio IDE.
2. Click on "Create new project."
3. In the "Create new project" window, select "ASP.NET Core Web Application" from the list of templates displayed.
4. Click Next.

5. In the "Configure your new project" window, specify the name and location for the new project.
6. Optionally, select the "Place solution and project in the same directory" check box.
7. Click Create.
8. In the "Create a New ASP.NET Core Web Application" window shown next, select .NET Core as the runtime and ASP.NET Core 2.2 (or later) from the drop-down list at the top. I'll be using **ASP.NET Core 3.0**.
9. Select "Web Application (Model-View-Controller)" as the project template to create a new ASP.NET Core MVC application.
10. Ensure that the check boxes "Enable Docker Support" and "Configure for HTTPS" are unchecked as we won't be using those features here.
11. Ensure that Authentication is set to "No Authentication" as we won't be using authentication either.
12. Click Create.

Following these steps should create a new ASP.NET Core MVC project in Visual Studio. We'll use this project to illustrate exporting data for Excel in the sections below.



Install the ClosedXML NuGet package

There are several UNITED STATES ▼ libraries to choose from if you want to export data to Excel. One of them is named ClosedXML. You can install this package either via the NuGet package manager inside the Visual Studio 2019 IDE, or by executing the following command in the NuGet package manager console:

```
Install-Package ClosedXML
```

Export data as a CSV file from ASP.NET Core 3.0

Exporting data as a comma-separated (CSV) file is simple. You could take advantage of a NuGet package such as CsvExport or AWright18.SimpleCSVExporter to achieve this, or you could do it manually. For the sake of simplicity, we'll generate a CSV file manually. Consider the following class named Author.


```
public class Author
{
    public int Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
}
```

Next, you can populate data into a list of authors as shown in the code snippet given below.

RECOMMENDED WHITEPAPERS

Best Cloud ERP Systems: Top 10 Picks & Vendor Checklist

Embedded BI Software Tools: Top 7 Pricing Guide

 How to Future-Proof your Field Service Technology Infrastructure

```

List<Author> authors = new List<Author>
{
    new Author { Id = 1, FirstName = "Joydip", LastName = "Kanjilal" },
    new Author { Id = 2, FirstName = "Steve", LastName = "Smith" },
    new Author { Id = 3, FirstName = "Anand", LastName = "Narayaswamy"}
};

```

The following code snippet shows how you can generate a CSV file in an action method of your controller.

```

public IActionResult DownloadCommaSeperatedFile()
{
    try
    {
        StringBuilder stringBuilder = new StringBuilder();
        stringBuilder.AppendLine("Id,FirstName,LastName");
        foreach (var author in authors)
        {
            stringBuilder.AppendLine($"{author.Id},
            {author.FirstName},{author.LastName}");
        }
        return File(Encoding.UTF8.GetBytes
        (stringBuilder.ToString()), "text/csv", "authors.csv");
    }
    catch
    {
        return Error();
    }
}

```

Export data as an XLSX file in ASP.NET Core 3.0

A workbook in Excel consists of several worksheets. You can create an Excel workbook using the following code.

```

var workbook = new XLWorkbook();

```

You can then take advantage of the `IXLWorkSheet` interface to create and add worksheets to the workbook as shown below.

```

IXLWorksheet worksheet = workbook.Worksheets.Add("Authors");
worksheet.Cell(1, 1).Value = "Id";
worksheet.Cell(1, 2).Value = "FirstName";
worksheet.Cell(1, 3).Value = "LastName";
for (int index = 1; index <= authors.Count; index++)
{
    worksheet.Cell(index + 1, 1).Value = authors[index - 1].Id;
    worksheet.Cell(index + 1, 2).Value = authors[index - 1].FirstName;
    worksheet.Cell(index + 1, 3).Value = authors[index - 1].LastName;
}

```

Lastly, you can save the workbook as a memory stream and then create a `FileContentResult` instance as shown below.

```

using (var stream = new MemoryStream())
{
    workbook.SaveAs(stream);
    var content = stream.ToArray();
    return File(content, contentType, fileName);
}

```

Download an Excel document in ASP.NET Core 3.0

Here is the complete source code of the action method that can be used to download an Excel document.

```

public IActionResult DownloadExcelDocument()
{
    string contentType = "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet";
    string fileName = "authors.xlsx";
    try
    {
        using (var workbook = new XLWorkbook())
        {
            IXLWorksheet worksheet =
                workbook.Worksheets.Add("Authors");
            worksheet.Cell(1, 1).Value = "Id";
            worksheet.Cell(1, 2).Value = "FirstName";
            worksheet.Cell(1, 3).Value = "LastName";
            for (int index = 1; index <= authors.Count; index++)
            {
                worksheet.Cell(index + 1, 1).Value =
                    authors[index - 1].Id;
                worksheet.Cell(index + 1, 2).Value =
                    authors[index - 1].FirstName;
                worksheet.Cell(index + 1, 3).Value =
                    authors[index - 1].LastName;
            }
            using (var stream = new MemoryStream())
            {
                workbook.SaveAs(stream);
                var content = stream.ToArray();
                return File(content, contentType, fileName);
            }
        }
    }
    catch (Exception ex)
    {
        return Error();
    }
}

```

While we have used ClosedXML in this article, there are several other packages for reading, writing, and manipulating Excel data in ASP.NET Core including EPPlus and NPOI. You can learn more about ClosedXML on GitHub at <https://github.com/ClosedXML/ClosedXML>. I'll discuss importing Excel data in an ASP.NET Core application in a future post here.

How to do more in ASP.NET and ASP.NET Core:

UNITED STATES ▼

- How to use in-memory caching in ASP.NET Core
- How to handle errors in ASP.NET Web API
- How to pass multiple parameters to Web API controller methods
- How to log request and response metadata in ASP.NET Web API
- How to work with HttpModules in ASP.NET
- Advanced versioning in ASP.NET Core Web API
- How to use dependency injection in ASP.NET Core
- How to work with sessions in ASP.NET
- How to work with HTTPHandlers in ASP.NET
- How to use IHostedService in ASP.NET Core
- How to consume a WCF SOAP service in ASP.NET Core
- How to improve the performance of ASP.NET Core applications
- How to consume an ASP.NET Core Web API using RestSharp
- How to work with logging in ASP.NET Core
- How to use MediatR in ASP.NET Core
- How to work with session state in ASP.NET Core
- How to use Nancy in ASP.NET Core
- Understand parameter binding in ASP.NET Web API
- How to upload files in ASP.NET Core MVC
- How to implement global exception handling in ASP.NET Core Web API
- How to implement health checks in ASP.NET Core
- Best practices in caching in ASP.NET
- How to use Apache Kafka messaging in .NET
- How to enable CORS on your Web API
- When to use WebClient vs. HttpClient vs. HttpWebRequest

- How to work with Redis Cache in .NET
- When to use Task.WaitAll vs. Task.WhenAll in .NET

Joydip Kanjilal is a Microsoft MVP in ASP.Net, as well as a speaker and author of several books and articles. He has more than 20 years of experience in IT including more than 16 years in Microsoft .Net and related technologies.

Follow     

Copyright © 2020 IDG Communications, Inc.

- Stay up to date with InfoWorld's newsletters for software developers, analysts, database programmers, and data scientists.
- Get expert insights from our member-only Insider articles.

YOU MAY ALSO LIKE

Recommended by

Tim O'Reilly: the golden age of the programmer is over

Python developers want static typing

Are industry clouds an opportunity or a distraction?

Amazon Braket: Get started with quantum computing

What's new in Kubernetes 1.20

When Kubernetes is not the solution

The decline of Heroku

Review: 7 Python IDEs go to the mat

What is a computational storage drive? Much-

The 24 highest paying developer roles in 2020

Google's Logica language addresses SQL's flaws

SPONSORED LINKS

dtSearch® instantly searches terabytes of files, emails, databases, web data. See site for hundreds of reviews; enterprise & developer evaluations

Truly modern web app and API security thinking. It's a thing. See how.

Want lightning fast analytics? See why the Incorta data analytics platform is changing enterprise data forever.

2020 was a year of rapid progression of digital transformation for businesses. The following is a snapshot of the digital transformation advancements made across all facets of business.

DDoS extortion attacks are real. Don't Negotiate. Mitigate with NETSCOUT. Learn more.

