**Swagger With .NET Core**

Swagger is a tool which lets you describe the structure of your APIs in a machine-readable format. Swagger achieves this by inspecting your APIs to generate a YAML or JSON file which contains a detailed description of your entire API.

Swagger implements the OpenAPI Specification and automates the entire documentation process, making it effortless for your teams to generate and maintain them. It works in conjunction with Swagger UI so it generates user-friendly and interactive API documentation

**Getting Started**

Create a new .NET Core 2.0 Web API project and install Swashbuckle.AspNetCore nuget package:

Package Manager

========================

PM> Install-Package Swashbuckle.AspNetCore -Version 3.0.0

.NET CLI

========================

> dotnet add package Swashbuckle.AspNetCore --version 3.0.0

The Swashbuckle.AspNetCore metapackage contains three components:

**Swashbuckle.AspNetCore.Swagger:**

A middleware to expose Swagger specification document as JSON endpoints from APIs

**Swashbuckle.AspNetCore.SwaggerGen:**

A Swagger Generator that inspects the controller, routes and models to generate Swagger specification document for APIs

**Swashbuckle.AspNetCore.SwaggerUI:**

A Swagger UI tool. It reads the Swagger JSON to build a rich UI for describing the Web API functionality. It also comes with a built-in test harnesses which lets users interact with APIs.

**Wiring Up**

*ConfigureServices* method of *Startup.cs* *class* to get everything going. First, add the Swagger Generator to the services collection.

// This method gets called by the runtime. Use this method to add services to the container.

public void ConfigureServices(IServiceCollection services)

{

services.AddMvc();

// Registering Swagger Generator

services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new Info

{

Title = "Swagger Demo",

Version = "v1",

Description = "TBD",

TermsOfService = "None",

Contact = new Contact() { Name = "John Doe", Email = "john@xyzmail.com", Url = "www.example.com" },

License = new License() { Name = "License Terms", Url = "www.example.com" }

});

});

}

Now, we need to enable the middleware for serving the generated JSON document and Swagger UI. To enable the middleware, update the *Configure* method of *Startup class*.

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

public void Configure(IApplicationBuilder app, IHostingEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

// Serves generated swagger document as JSON endpoint.

app.UseSwagger();

// Serves the Swagger UI

app.UseSwaggerUI(c =>

{

// specifying the Swagger JSON endpoint.

c.SwaggerEndpoint("/swagger/v1/swagger.json", "Swagger Demo");

});

app.UseMvc();

}

The *UseSwagger* method serves the generated JSON document at http://localhost:<port>/swagger/v1/swagger.json

The *UseSwaggerUI* method serves the Swagger UI at http://localhost:<port>/swagger

The Swagger UI can also be served at the app’s root (http://localhost:<port>/), for that set the *RoutePrefix*property to an empty string.

// Serves the Swagger UI

app.UseSwaggerUI(c =>

{

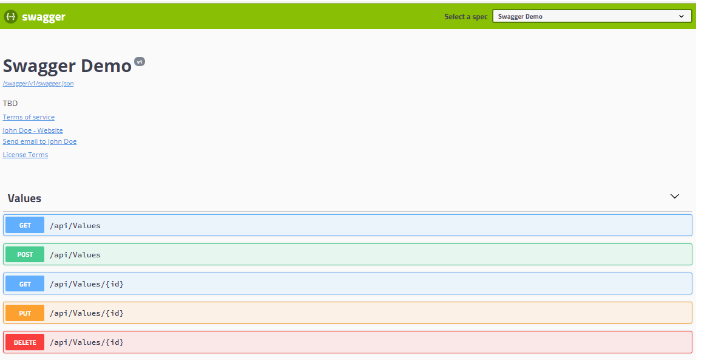
// specifying the Swagger JSON endpoint.

c.SwaggerEndpoint("/swagger/v1/swagger.json", "Swagger Demo");

c.RoutePrefix = string.Empty;

});

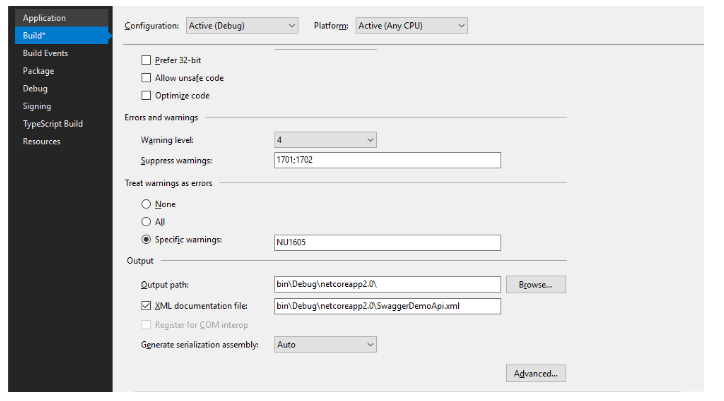
Now run the application and navigate to the application root. You should see the output like the one shown below:



**Enabling XML comments:**

Here’s how you enable XML comments: right click on the Web API project and select **properties**, then navigate to the Build tab. Under **Output** check the **XML documentation file**box. This will generate the XML documentation file for the Debug mode.

For release mode change **Configuration** to **Release** and again check the **XML documentation file**box.



Now, we will configure Swagger to consume this file. To do so, we need to pass this file to *AddSwaggerGen*method. Update the Swagger generator registration as shown below:

// Registering Swagger Generator

services.AddSwaggerGen(c =>

{

...

// Configure the XML comments file path for the Swagger JSON and UI.

var xmlFile = $"{System.Reflection.Assembly.GetExecutingAssembly().GetName().Name}.xml";

c.IncludeXmlComments(System.IO.Path.Combine(AppContext.BaseDirectory, xmlFile));

});

The generated XML documentation file has the same name as that of the Web API project. The AppContext.BaseDirectory property gets the location of the XML file.

Now it’s time to add comments to the action methods. To add triple slash comments type 3 forward slashes above the method name and visual studio will auto-generate the comment section. Add the comments between <summary> tags and run the solution. The comments will be displayed in UI as shown below:

/// <summary>

/// Deletes individual item by id.

/// </summary>

/// <param name="id">ID Value of the item</param>

// DELETE api/employees/5

[HttpDelete("{id}")]

public void Delete(int id)

{

// Delete implementation

}

You can also add more robust comments. Just add a *<remarks>* section in the comments as described below

/// <summary>

/// Creates a new employee

/// </summary>

/// <param name="employee">Employee Object</param>

/// <param name="employeeType"></param>

/// <remarks>

/// Sample request:

///

/// POST /employee

/// {

/// "id": 007,

/// "name": "James Bond",

/// "isPermanant": true

/// }

///

/// </remarks>

// POST api/employees

[HttpPost]

public void Post([FromBody]Employee employee, EmployeeType employeeType)

{

// Create employee implementaion

}

**Display enum as strings**

By default, Swagger displays enums as integers in Swagger UI. In order to display enum values as a strings add *DescribeAllEnumsAsStrings* to *AddSwaggerGen*method in*Startup*class. To display enum string values in camel case use *DescribeStringEnumsInCamelCase*.

// Registering Swagger Generator

services.AddSwaggerGen(c =>

{

c.DescribeAllEnumsAsStrings();

c.DescribeStringEnumsInCamelCase();

...

});

**Describing response types**

Swagger generates a “200” response of all operations. If any action method returns a model then it will be utilized to generate a schema for the response body.

In certain instances, we may want to return different response codes for the operations performed by the action methods. We can describe those responses in the Swagger documentation by decorating the corresponding action methods with *ProducesResponseType* attribute.

// GET api/employees/5

[HttpGet("{id}")]

[ProducesResponseType(typeof(Employee), 200)]

[ProducesResponseType(typeof(IDictionary<string, string>), 400)]

[ProducesResponseType(500)]

public IActionResult Get(Employee id)

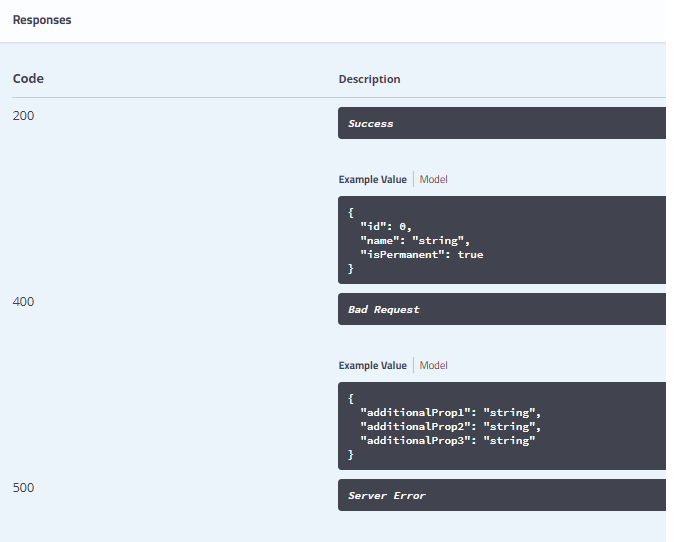
{

// Get employee implementation

...

return Ok(new Employee());

}



We can also customize the description of a response codes by adding the response tag in the comments section above action method.

// GET api/employees/5

/// <response code="200">Request completed Successfully</response>

[HttpGet("{id}")]

[ProducesResponseType(typeof(Employee), 200)]

[ProducesResponseType(typeof(IDictionary<string, string>), 400)]

[ProducesResponseType(500)]

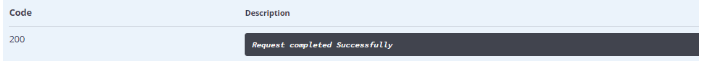
public IActionResult Get(Employee id)

{

// Get employee implementation

return Ok(new Employee());

}



**Multiple API Versions**

Swagger supports API versioning. Using Swagger, we can create multiple documentations for the different versions of API. Adding versioning to Swagger requires a couple of updates in the *Startup* class. We need to add an additional Swagger document using *SwaggerDoc*for version*‘v2’.*Then configure the same for Swagger UI using *SwaggerEndpoint.*

public void ConfigureServices(IServiceCollection services)

{

...

services.AddApiVersioning(o => o.ApiVersionReader = new HeaderApiVersionReader("api-version"));

// Registering Swagger Generator

services.AddSwaggerGen(c =>

{

...

c.SwaggerDoc("v1", new Info

{

Title = "Swagger Demo",

Version = "v1",

Description = "TBD",

TermsOfService = "None",

Contact = new Contact() { Name = "John Doe", Email = "john@xyzmail.com", Url = "www.example.com" },

License = new License() { Name = "License Terms", Url = "www.example.com" }

});

c.SwaggerDoc("v2", new Info

{

Title = "Swagger Demo for v2",

Version = "v2",

Description = "TBD",

TermsOfService = "None",

Contact = new Contact() { Name = "John Doe", Email = "john@xyzmail.com", Url = "www.example.com" },

License = new License() { Name = "License Terms", Url = "www.example.com" }

});

...

});

}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

public void Configure(IApplicationBuilder app, IHostingEnvironment env)

{

...

// Serves the Swagger UI

app.UseSwaggerUI(c =>

{

// specifying the Swagger JSON endpoint.

c.SwaggerEndpoint("/swagger/v1/swagger.json", "Swagger Demo");

c.SwaggerEndpoint("/swagger/v2/swagger.json", "Swagger Demo for v2");

});

...

}

By default, Swashbuckle will include all API endpoints in both documents, so we will have to inform Swashbuckle which API endpoints to include in each document. This can be achieved either by decorating Individual Actions or by applying an application wide convention.

In case of decorating individual actions, we will have to decorate each action with the ApiExplorerSettingsAttribute and set GroupName to the corresponding document name (case sensitive):

// POST api/employees

[HttpPost]

[ApiExplorerSettings(GroupName = "v2")]

public void Post([FromBody]Employee employee, EmployeeType employeeType)

{

// Create employee implementaion

}

**Customizing Swagger UI**

The look and feel of the default UI that’s shipped with Swagger UI package can be customized by injecting custom CSS stylesheets. For customizing the Swagger UI, create a new CSS file *(custom.css)* in the wwwroot/css folder of the API project. Now, inject it in the middleware with *InjectStylesheet*method.

.swagger-ui .topbar {

background-color: #000;

}

.swagger-ui label {

color: #FFF;

}

.swagger-ui .topbar .download-url-wrapper .select-label select {

border: 2px solid #aaa;

}

// Serves the Swagger UI

app.UseSwaggerUI(c =>

{

...

c.InjectStylesheet("/css/custom.css");

});



In the same way custom Javascript code can also be injected using *InjectJavascript*method in Swagger UI.

**Generating Swagger Example Request and Response:**

First, install the .NET Core version [Swashbuckle.AspNetCore.Examples](https://www.nuget.org/packages/Swashbuckle.AspNetCore.Examples/) From the nuget

Now decorate your controller methods with the included SwaggerRequestExample attribute:

[SwaggerRequestExample(typeof(DeliveryOptionsSearchModel), typeof(DeliveryOptionsSearchModelExample))]

[SwaggerResponseExample(HttpStatusCode.OK, typeof(DeliveryOptionsModelExample))]

public async Task<IHttpActionResult> DeliveryOptionsForAddress(DeliveryOptionsSearchModel search)

{

Now implement it, in this case via a DeliveryOptionsSearchModelExample (which should implement IExamplesProvider), which will generate the example data. It should return the type you specified when you specified the [SwaggerRequestExample].

public class DeliveryOptionsSearchModelExample : IExamplesProvider

{

    public object GetExamples()

    {

        return new DeliveryOptionsSearchModel

        {

            Lang = "en-GB",

            Currency = "GBP",

            Address = new AddressModel

            {

                Address1 = "1 Gwalior Road",

                Locality = "London",

                Country = "GB",

                PostalCode = "SW15 1NP"

            },

            Items = new[]

            {

                new ItemModel

                {

                    ItemId = "ABCD",

                    ItemType = ItemType.Product,

                    Price = 20,

                    Quantity = 1,

                    RestrictedCountries = new[] { "US" }

                }

            }

        };

    }

Don’t forget to enable the ExamplesOperationFilter when you enable Swagger, as before:

services.AddSwaggerGen(c =>

   {

        c.SwaggerDoc("v1", new Info { Title = "My API", Version = "v1" });

        c.OperationFilter<ExamplesOperationFilter>();

**Omit the methods from the swagger Documentation**

You can add the following attribute to Controllers and Actions to exclude them from the generated documentation: [ApiExplorerSettings(IgnoreApi = true)]

[**Using swagger/swashbuckle .net core operation filters to enable API testing**](http://boris-zaikin.blogspot.com/2017/09/using-swaggerswashbuckle-net-core.html)

The problem arises in solutions which contains identity servers based on OAuth2 and OpenId protocols (For example [Identity Server](http://docs.identityserver.io/en/release/), [WSO2 identity cloud](https://wso2.com/identity-and-access-management#cloud), etc.) and Rest API with swagger [swashbuckle](https://github.com/domaindrivendev/Swashbuckle) used for auto documenting and testing. Each API call should contain authentication token otherwise testing API ("Try it out") will not work.

**Solution**

One of the option to enable API testing ("Try it out" button) is using swagger [operation filter](https://docs.microsoft.com/en-us/azure/app-service-api/app-service-api-dotnet-swashbuckle-customize). In example below was created operation filter which includes Bearer authentication token field to HTTP header of each API calls.

public class AuthorizationHeaderOperationFilter : IOperationFilter

{

public void Apply(Operation operation, OperationFilterContext context)

{

IList<FilterDescriptor> filterDescriptors = context.ApiDescription.ActionDescriptor.FilterDescriptors;

bool isAuthorized = filterDescriptors.Select(filterInfo => filterInfo.Filter).Any(filter => filter is AuthorizeFilter);

bool allowAnonymous = filterDescriptors.Select(filterInfo => filterInfo.Filter).Any(filter => filter is IAllowAnonymousFilter);

if (isAuthorized && !allowAnonymous)

{

if (operation.Parameters == null)

{

operation.Parameters = new List<IParameter>();

}

operation.Parameters.Add(new NonBodyParameter

{

Name = "Authorization",

In = "header",

Description = "access token",

Required = true,

Type = "string",

Default = "Bearer "

});

}

}

}

Then filter should included in swagger [swashbuckle](https://github.com/domaindrivendev/Swashbuckle) configuration in ConfigureServices method of Startup.cs file

services.AddSwaggerGen(setupAction =>

{

setupAction.SwaggerDoc("v2", new Info { Title = "DeviceAPI", Version = $"v2", Description = "The Device API" });

// Filter Registration

setupAction.OperationFilter<AuthorizationHeaderOperationFilter>();

});

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Quick summary** | **Source (specification)** | **Live demo** |
| **Swagger** | Whole ecosystem, lots of integrations Good-looking UI for docs Widely used, many resources available | Swagger/OpenAPI | [**Swagger demo**](http://petstore.swagger.io/) |
| **DapperDox** | Inject relevant documentation right into the rendered specification page | OpenAPI, Markdown | [**DapperDox demo**](http://dapperdox.io/docs/overview) |
| **ReDoc** | Easy deployment Wide support for OpenAPI objects Interactive, responsive documentation | OpenAPI | [**ReDoc demo**](https://rebilly.github.io/RebillyAPI/) |
| **RAML 2 HTML** | Simple RAML to HTML documentation generator theme support | RAML, NodeJSwith | [**RAML 2 HTML demo**](https://rawgit.com/raml2html/default-theme/master/examples/helloworld.html) |
| **RAML API Console** | Browsing of API documentation and in-browser testing of API methods | RAML, NodeJS | [**RAML API Console demo**](https://anypoint.mulesoft.com/apiplatform/popular/#/portals/organizations/52560d3f-c37a-409d-9887-79e0a9a9ecff/apis/5502/versions/5487/pages/30295) |
| **Snowboard** | API Blueprint renderer | API Blueprint | [**Snowboard demo**](https://htmlpreview.github.io/?https://github.com/subosito/snowboard/blob/master/examples/alpha/Real%20World%20API.html) |
| **Aglio** | API Blueprint renderer with many custom themes | API Blueprint | [**Aglio demo**](http://htmlpreview.github.io/?https://raw.githubusercontent.com/danielgtaylor/aglio/blob/master/examples/cyborg.html) |
| **I/O Docs** | Live interactive API documentation system for I/O Docs specification format | I/O Docs (JSON) | [**I/O Docs demo**](http://support.mashery.com/io-docs) |
| **Slate** | Clean, intuitive design Write in Markdown Collaboration through GitHub | Markdown (Ruby) | [**Slate demo**](https://lord.github.io/slate/) |
| **Whiteboard** | NodeJS based Slate alternative | NodeJS | [**Whiteboard demo**](https://wifidistribution.com/docs/) |
| **apiDoc** | Inline documentation for RESTful web APIs | NodeJS | [**apiDoc demo**](http://apidocjs.com/example/) |
| **CuuBEZ API Visualizer** | Visualize the documentation of RESTful web APIs | Java | [**CuuBEZ API Visualizer demo**](http://apivisualizerold.cuubez.com/demo.html) |
| **Apidox** | XML powered live interactive API documentation and browsing for RESTful APIs | XML, PHP | [**Apidox demo**](http://apidox.net/demo/apidox.php) |
| **Carte** | A simple Jekyll based documentation website  for APIs | Jekyll, YAML | [**Carte demo**](http://wiredcraft.github.io/carte/) |
| **Docbox** | A responsive website generated from Markdown documentation content | Markdown | [**Docbox demo**](https://50-53007065-gh.circle-artifacts.com/0/tmp/circle-artifacts.8SMOD8H/index.html#our-api) |
| **API Docs** | Free, hosted API documentation | OpenAPI, Swagger, RAML | [**API Docs demo**](https://giphy.api-docs.io/1.0/welcome) |