Swagger (OpenAPI) Specification code generator featuring C# and Razor templates. Supports C#, Java, Node.js, TypeScript, Python and Ruby.

code-generator csharp golang java javascript node openapi python ruby swagger

3,730 commits 20 branches 49 releases 67 contributors MIT

sarangan12 committed on GitHub Code changes to generate resources/subresources (#1769) … Latest commit 1fae9ec 7 hours ago

PackageTest	Promote RefactorCodeModel to master (#1545)	3 months ago
Samples	cleaner code gen (less unnecessary code) (#1771)	a day ago
Tools	[Java] AutoRest changes by runtime changes (#1703)	9 days ago
docs	Literate File Formats (#1743)	2 days ago
schema	added operationId and title as optional properties of the example sch…	15 days ago
src	Code changes to generate resources/subresources (#1769)	7 hours ago
.gitattributes	Fixing C# simplifier, and line endings in generated code. (#1627)	3 months ago
.gitignore	Generate Server side code from specs (#1697)	27 days ago
.ruby-version	add in a root level gemfile so that we will have all ruby test depend…	2 years ago
.travis.yml	[Java] AutoRest changes by runtime changes (#1703)	9 days ago
AutoRest.sln	CSharp.Azure.Fluent refactor and some Go fixes (#1705)	28 days ago
AutoRest.sln.DotSettings	Promote RefactorCodeModel to master (#1545)	3 months ago
ChangeLog.md	Adding release notes for 0.16.0 release	9 months ago
Gemfile	Updated gemfiles to support faraday.	2 years ago
Issues.md	Page with links to Issues Queries	8 months ago
LICENSE	update copyright year	6 months ago
README.md	Improvements for Mono & Docker users (#1714)	20 days ago
build.proj	Generator should not flatten out parameters if they are polymorphic (#…	15 days ago
global.json	Moving Clientruntimes out of autorest	4 months ago
gulpfile.js	[Java] AutoRest changes by runtime changes (#1703)	9 days ago
package.json	Bumped up gulp npm versions	8 months ago
pom.xml	[Java] AutoRest changes by runtime changes (#1703)	9 days ago

README.md

build passing Issue Stats Issue Stats

AutoRest

The **AutoRest** tool generates client libraries for accessing RESTful web services. Input to *AutoRest* is a spec that describes the REST API using the Open API Initiative format.

Getting AutoRest

The AutoRest tools can be installed with Nuget for use in a Visual Studio project: AutoRest NuGet

Alternatively it can be installed from Chocolatey by running: chocolatey v0.16.0

```
choco install autorest
```

Nightlies are available via MyGet: AutoRest MyGet

AutoRest can be run on macOS and *nix using Mono:

Download & Unpack Autorest curl -LO https://github.com/Azure/autorest/releases/download/AutoRest-0.16.0/autorest.0.16.0.zip && \ unzip autorest.0.16.0.zip -d autorest/ && \ cd autorest && \

Download Swagger.json example curl -O https://raw.githubusercontent.com/Azure/autorest/master/Samples/petstore/petstore.json && \

Run AutoRest using mono mono AutoRest.exe \ -CodeGenerator CSharp \ -Input petstore.json \ -OutputDirectory CSharp_PetStore -Namespace PetStore

Or Docker:

Download Swagger.json example curl -O https://raw.githubusercontent.com/Azure/autorest/master/Samples/petstore/petstore.json

Download latest AutoRest Docker image docker pull azuresdk/autorest:latest

Run AutoRest using Docker, mounting the current folder (pwd) into /home inside the container docker run -it --rm -v $(pwd):/home azuresdk/autorest:latest autorest \ -CodeGenerator CSharp \ -Input /home/petstore.json \ -OutputDirectory /home/CSharp_PetStore -Namespace PetStore

Building AutoRest

AutoRest is developed primarily in C# but generates code for multiple languages. See this link to build and test AutoRest.

> Hint: There is a powershell script (`verify-settings.ps1`) in the `Tools` folder that can verify that you have the required compilers/tools/libraries installed on your development system before trying to build.

Hello World

For this version of Hello World, we will use **AutoRest** to generate a client library and use it to call a web service. The trivial web service that just returns a string is defined as follows:

```
public class HelloWorldController : ApiController
{
    // GET: api/HelloWorld
    public string Get()
    {
        return "Hello via AutoRest.";
    }
}
```

By convention, Swagger documents are exposed by web services with the name `swagger.json` . The `title` property of the `info` object is used by **AutoRest** as the name of the client object in the generated library. The `host + path` of the operation corresponds to the URL of the operation endpoint. The `operationId` is used as the method name. The spec declares that a `GET` request will return an HTTP 200 status code with content of mime-type `application/json` and the body will be a string. For a more in-depth overview of swagger processing, refer to Defining Clients With Swagger section of the documentation.

```
{
  "swagger": "2.0",
  "info": {
    "title": "MyClient",
    "version": "1.0.0"
  },
```

```
  "host": "swaggersample.azurewebsites.net",
  "paths": {
    "/api/HelloWorld": {
      "get": {
        "operationId": "GetGreeting",
        "produces": [
          "application/json"
        ],
        "responses": {
          "200": {
            "description": "GETs a greeting.",
            "schema": {
              "type": "string"
            }
          }
        }
      }
    }
  }
}
```

Next, we invoke **AutoRest.exe** with this swagger document to generate client library code (see Command Line Interface documentation for details).

AutoRest is extensible and can support multiple types of input and output. *AutoRest.exe* comes with the *AutoRest.json* configuration file that defines the available inputs (*Modelers*) and outputs (*CodeGenerators*). When invoking *AutoRest.exe*, if you don't specify the `-Modeler` then Swagger is assumed and if you don't specify `-CodeGenerator` then CSharp is used.

The Swagger schema is language agnostic and doesn't include the notion of namespace, but for generating code, AutoRest requires `-Namespace` be specified. By default, the CodeGenerator will place output in a directory named *Generated*. This can be overridden by providing the `-OutputDirectory` parameter.

```
AutoRest.exe -CodeGenerator CSharp -Modeler Swagger -Input swagger.json -Namespace MyNamespace
```

Now, we will use the generated code to call the web service.

Create a console application called *HelloWorld*. Add the generated files to it. They won't compile until you add the NuGet package the generated code depends on: `Microsoft.Rest.ClientRuntime` .

You can add it to the Visual Studio project using the NuGet package manager or in the Package Manager Console with this command:

```
Install-Package Microsoft.Rest.ClientRuntime
```

Add the namespace that was given to AutoRest.

```
using MyNamespace;
```

Access the REST API with very little code (see Client Initialization and Client Operations for details).

```
var myClient = new MyClient();
var salutation = myClient.GetGreeting();
Console.WriteLine(salutation);
```

Running the console app shows the greeting retrieved from the service API.

```
C:\>HelloWorld.exe
Hello via AutoRest.
```

With that same basic pattern in place, you can now explore how different REST API operations and payloads are described in Swagger and exposed in the code generated by **AutoRest**.

This project has adopted the Microsoft Open Source Code of Conduct. For more information see the Code of Conduct FAQ or contact opencode@microsoft.com with any additional questions or comments.