
Swagger to JS & Typescript Codegen

npm v1.13.0 npm v1.13.0

We are looking for a new maintainer

This project is no longer actively maintained by its creator. Please let us know if you would like to become a maintainer. At the time we wrote this package, the swagger didn't have generators for JavaScript nor TypeScript. Now there are great alternatives of this package available.

This package generates a nodejs, reactjs or angularjs class from a swagger specification file. The code is generated using mustache templates and is quality checked by jshint and beautified by js-beautify.

The typescript generator is based on superagent and can be used for both nodejs and the browser via browserify/webpack.

Installation

```
1 npm install swagger-js-codegen
```

Example

```
1 var fs = require('fs');
2 var CodeGen = require('swagger-js-codegen').CodeGen;
3
4 var file = 'swagger/spec.json';
5 var swagger = JSON.parse(fs.readFileSync(file, 'UTF-8'));
6 var nodejsSourceCode = CodeGen.getNodeCode({ className: 'Test', swagger
: swagger });
7 var angularjsSourceCode = CodeGen.getAngularCode({ className: 'Test',
swagger: swagger });
8 var reactjsSourceCode = CodeGen.getReactCode({ className: 'Test',
swagger: swagger });
9 var tsSourceCode = CodeGen.getTypescriptCode({ className: 'Test',
swagger: swagger, imports: ['../../typings/tsd.d.ts'] });
10 console.log(nodejsSourceCode);
11 console.log(angularjsSourceCode);
12 console.log(reactjsSourceCode);
13 console.log(tsSourceCode);
```

Custom template

```
1 var source = CodeGen.getCustomCode({
2   moduleName: 'Test',
3   className: 'Test',
4   swagger: swaggerSpec,
5   template: {
6     class: fs.readFileSync('my-class.mustache', 'utf-8'),
7     method: fs.readFileSync('my-method.mustache', 'utf-8'),
8     type: fs.readFileSync('my-type.mustache', 'utf-8')
9   }
10 });
```

Options

In addition to the common options listed below, `getCustomCode()` requires a `template` field:

```
1 template: { class: "...", method: "..." }
```

`getAngularCode()`, `getNodeCode()`, and `getCustomCode()` each support the following options:

```
1  moduleName:
2    type: string
3    description: Your AngularJS module name
4  className:
5    type: string
6  lint:
7    type: boolean
8    description: whether or not to run jslint on the generated code
9  esnext:
10    type: boolean
11    description: passed through to jslint
12  beautify:
13    type: boolean
14    description: whether or not to beautify the generated code
15  mustache:
16    type: object
17    description: See the 'Custom Mustache Variables' section below
18  imports:
19    type: array
20    description: Typescript definition files to be imported.
21  swagger:
22    type: object
23    required: true
24    description: swagger object
```

Template Variables

The following data are passed to the mustache templates:

```
1  isNode:
2    type: boolean
3  isES6:
4    type: boolean
5  description:
6    type: string
7    description: Provided by your options field: 'swagger.info.
      description'
8  isSecure:
9    type: boolean
10   description: false unless 'swagger.securityDefinitions' is defined
11  moduleName:
12    type: string
13    description: Your AngularJS module name - provided by your options
      field
14  className:
15    type: string
16    description: Provided by your options field
17  domain:
18    type: string
19    description: If all options defined: swagger.schemes[0] + '://' +
      swagger.host + swagger.basePath
20  methods:
21    type: array
22    items:
23      type: object
24      properties:
25        path:
26          type: string
27        className:
28          type: string
29          description: Provided by your options field
30        methodName:
31          type: string
32          description: Generated from the HTTP method and path elements
            or 'x-swagger-js-method-name' field
33        method:
34          type: string
35          description: 'GET', 'POST', 'PUT', 'DELETE', 'PATCH', 'COPY', '
            HEAD', 'OPTIONS', 'LINK', 'UNLIK', 'PURGE', 'LOCK', 'UNLOCK'
            , 'PROPFIND'
36        enum:
37          - GET
38          - POST
39          - PUT
40          - DELETE
41          - PATCH
```

```
42         - COPY
43         - HEAD
44         - OPTIONS
45         - LINK
46         - UNLIK
47         - PURGE
48         - LOCK
49         - UNLOCK
50         - PROPFIND
51     isGET:
52         type: string
53         description: true if method === 'GET'
54     summary:
55         type: string
56         description: Provided by the 'description' or 'summary' field
57                     in the schema
58     externalDocs:
59         type: object
60         properties:
61             url:
62                 type: string
63                 description: The URL for the target documentation. Value
64                             MUST be in the format of a URL.
65                 required: true
66             description:
67                 type: string
68                 description: A short description of the target
69                             documentation. GitHub-Markdown syntax can be used for
70                             rich text representation.
71     isSecure:
72         type: boolean
73         description: true if the 'security' is defined for the method
74                     in the schema
75     parameters:
76         type: array
77         description: Includes all of the properties defined for the
78                     parameter in the schema plus:
79         items:
80             camelCaseName:
81                 type: string
82             isSingleton:
83                 type: boolean
84                 description: true if there was only one 'enum' defined for
85                             the parameter
86             singleton:
87                 type: string
88                 description: the one and only 'enum' defined for the
89                             parameter (if there is only one)
90             isBodyParameter:
91                 type: boolean
92             isPathParameter:
```

```

85         type: boolean
86     isQueryParameter:
87         type: boolean
88     isPatternType:
89         type: boolean
90         description: true if *in* is 'query', and 'pattern' is
                       defined
91     isHeaderParameter:
92         type: boolean
93     isFormParameter:
94         type: boolean

```

Custom Mustache Variables You can also pass in your own variables for the mustache templates by adding a `mustache` object:

```

1  var source = CodeGen.getCustomCode({
2      ...
3      mustache: {
4          foo: 'bar',
5          app_build_id: env.BUILD_ID,
6          app_version: pkg.version
7      }
8  });

```

Swagger Extensions

x-proxy-header

Some proxies and application servers inject HTTP headers into the requests. Server-side code may use these fields, but they are not required in the client API.

eg: https://cloud.google.com/appengine/docs/go/requests#Go_Request_headers

```

1  /locations:
2      get:
3          parameters:
4              - name: X-AppEngine-Country
5                in: header
6                x-proxy-header: true
7                type: string
8                description: Provided by AppEngine eg - US, AU, GB
9              - name: country
10                 in: query
11                 type: string
12                 description: |
13                     2 character country code.

```

```
14         If not specified, will default to the country provided in the
15         X-AppEngine-Country header
16     ...
```

Grunt task

There is a grunt task that enables you to integrate the code generation in your development pipeline. This is extremely convenient if your application is using APIs which are documented/specified in the swagger format.

Who is using it?

28.io is using this project to generate their nodejs and angularjs language bindings.