

**Documentation on Swagger**

**Created By: Santosh D**  
 **Reviewed By: Sreenivas P**

### **What Is Open API Specification?**

Open API Specification (formerly Swagger Specification) is an API description format for REST APIs. An Open API file allows you to describe your entire API, including:

* Available endpoints (/users) and operations on each endpoint (GET /users, POST /users)
* Operation parameters Input and output for each operation
* Authentication methods
* Contact information, license, terms of use and other information.

API specifications can be written in YAML or JSON. The format is easy to learn and readable to both humans and machines.

**What is YAML?**

**YAML** (a recursive acronym for "**YAML** ain't Markup Language") is a human-readable data-serialization language. It is commonly used for configuration files and in applications where data is being stored or transmitted.

**What is JSON?**

JavaScript Object Notation is an open standard file format, and data interchange format, that uses human-readable text to store and transmit data objects consisting of attribute–value pairs

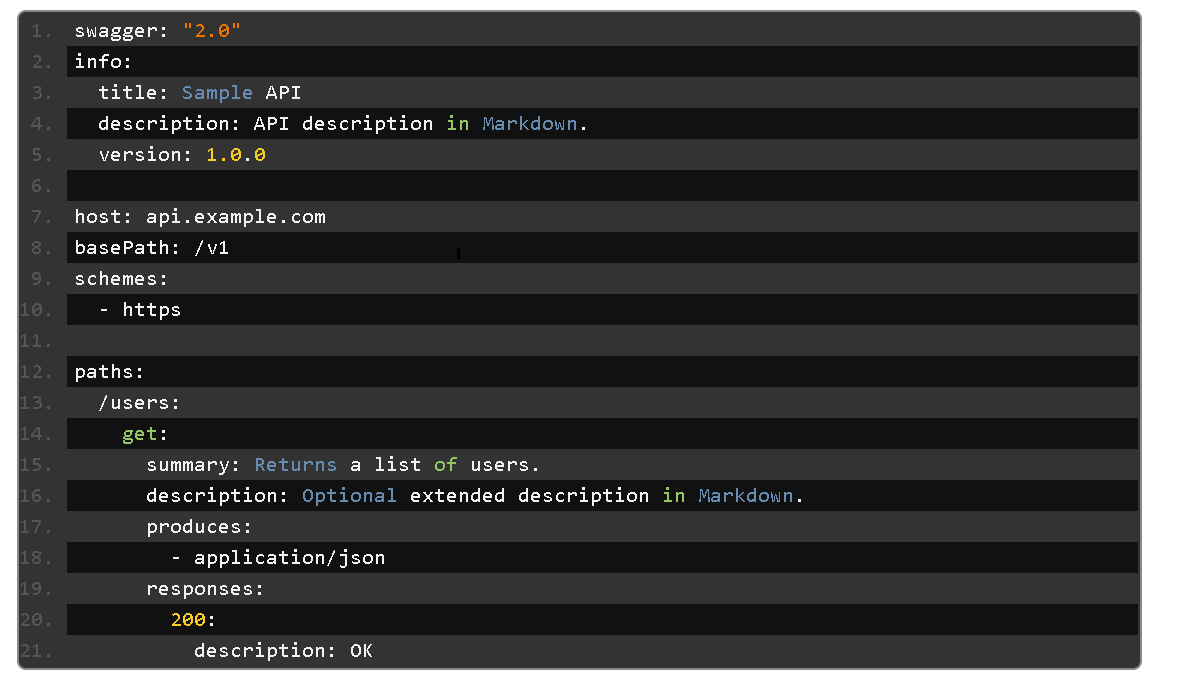
**What is Swagger?**

It is an OPEN Source API Documentation framework to help developers in design and document API’s.

* **Swagger** is a set of rules (in other words, a specification) for a format describing REST **APIs**.

[Swagger Editor](http://editor.swagger.io/) – browser-based editor where you can write Open API specs.

Basic Structure:



**Metadata:**

Every Swagger specification starts with the Swagger version, 2.0 being the latest version. A Swagger version defines the overall structure of an API specification – what you can document and how you document it.

**swagger: "2.0"**

Then, you need to specify the API info – title, description (optional), version (API version, not file revision or Swagger version).

**info:**  
 **title: Sample API**  
 **description: API description in Markdown.**  
 **version: 1.0.0**

### 

### **Base URL:** The base URL for all API calls is defined using schemes, host and base Path: **host: api.example.com** **Base ---Path: /v1** **schemes:** **- https**

**Consumes, Produces:**

The consumes and produces sections define the MIME types supported by the API. The root-level definition can be overridden in individual operations.

**consumes:**  
 **- application/json**  
 **- application/xml**

**produces:**  
 **- application/json**  
 **- application/xml**

**Paths:**

The paths section defines individual endpoints (paths) in your API, and the HTTP methods (operations) supported by these endpoints. For example, GET /users can be described as:

**paths:**  
**/users:**  
 **get:**  
 **summary: Returns a list of users.**  
 **description: Optional extended description in Markdown.**  
 **produces:**  
 **- application/json**  
 **responses:200**  
 **description: OK**

**Parameters:**

Operations can have parameters that can be passed via URL path (/users/{userId}), query string (/users?role=admin), headers (X-CustomHeader: Value) and request body. You can define the parameter types, format, whether they are required or optional, and other details:

**paths:**  
 **/users/{userId}:**  
 **get:**  
 **summary: Returns a user by ID.**  
 **parameters:**  
 **- in: path**  
 **name: userId**

**required: true**  
 **type: integer**  
 **minimum: 1**  
 **description: Parameter description in Markdown.**  
 **responses:**   
 **200:**  
 **description: OK**

**Responses:**

For each operation, you can define possible status codes, such as 200 OK or 404 Not Found, and schema of the response body. Schemas can be defined inline or referenced from an external definition via $ref. You can also provide example responses for different content types.

**paths:**

**/users/{userId}:**  
 **get:**  
 **summary: Returns a user by ID.**  
 **parameters:**  
 **- in: path**  
 **name: userId**  
 **required: true**  
 **type: integer**  
 **minimum: 1**

**description: The ID of the user to return.**

**responses:**  
 **200:**  
 **description: A User object.**  
 **schema:**  
 **type: object**  
 **properties:**  
 **id:**  
 **type: integer**  
 **example: 4**

**name:**  
 **type: string**  
 **example: Arthur Dent**  
 **400:**  
 **description: The specified user ID is invalid (e.g. not a number).**

**404:**  
 **description: A user with the specified ID was not found.**  
 **default:**  
 **description: Unexpected error**

**Input and Output Models:**

The global definitions section lets you define common data structures used in your API. They can be referenced via $ref whenever a schema is required – both for request body and response body. For example, this JSON object:

**{**

**"id": 4,**

**"name": "Arthur Dent"**

**}**

can be represented as:

**definitions:**  
**User:**  
 **properties:**  
 **id:**  
 **type: integer**  
 **name:**  
 **type: string**

**# Both properties are required**

**required:**   
 **- id**  
 **- name**

and then referenced in the request body schema and response body schema as follows:  
**paths:**  
**/users/{userId}:**  
 **get:**  
 **summary: Returns a user by ID.**  
 **parameters:**  
 **- in: path**

**name: userId**  
 **required: true**  
 **type: integer**

**responses:**  
 **200:**  
 **description: OK**  
 **schema:**  
 **$ref: '#/definitions/User'**  
 **/users:**  
 **post:**

**summary: Creates a new user.**  
**parameters:**  
 **- in: body**  
 **name: user**  
 **schema:**  
 **$ref: '#/definitions/User'**  
**responses:**  
**200:**  
**description: OK**

**Authentication:**

The security Definitions and security keywords are used to describe the authentication methods used in your API.

**Security Definitions:**  
  
 **Basic Auth:**  
 **type: basic**

**security:**  
  
**- Basic Auth: []**

Supported authentication methods are:

Basic authentication

API key (as a header or query parameter)

OAuth 2 common flows (implicit, password, application and access code)

**Reference Links:**

<https://swagger.io/docs/>

------------ 0 -----------