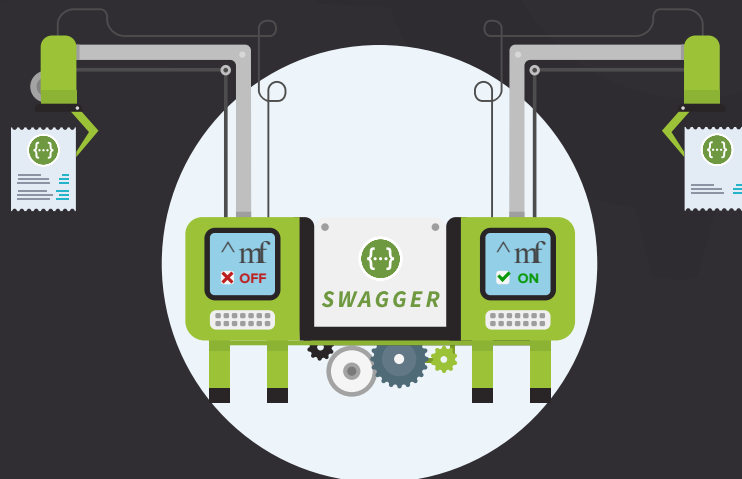


# DOCUMENTING AN EXISTING API WITH SWAGGER

---



Good user experience is key to using any product, and the same holds true for APIs. The better the interface that's used to consume APIs, the higher the chance of achieving your business and technological objectives.

In this whitepaper, we will be discussing documenting existing APIs using the Swagger framework. We will briefly walk through the different ways to generate the Swagger interface from an already coded API, and give an overview of adding documentation to the generated Swagger specification using SwaggerHub.

## Contents

The Importance of API Documentation	2
Why Use Swagger for Documentation	3
Approaches to Adding Swagger to Your APIs	5
Documenting Your Swagger Specification	7
Additional Resources	9

# An API is Only as Good as Its Documentation

APIs were conceptualized long before the advent of the PC, but were never openly discussed, and instead left to the shadows of an application's backend. APIs have only recently gained mass acceptance, starting with the prominent web-based API released by Salesforce in 2000. Since the advent of mobile and cloud computing, APIs have gone mainstream, with more and more companies and organizations understanding the business value of creating APIs. With a lot of web services emerging, the need to have clear API documentation for adopting these services became clear.

API documentation is the information that is required to successfully consume and integrate with an API. This would be in the form of technical writing, code samples and examples for better understanding how to consume an API. Concise and clear documentation — which allows your API consumers to adopt it into their application quickly — is no longer optional for organizations that want to drive adoption of their APIs.

## The importance of API documentation

A survey by ProgrammableWeb found that API consumers considered complete and accurate documentation as the biggest factor that figured in their API decision making, even outweighing price and API performance. Good documentation accelerates development and consumption, and reduces the money and time that would otherwise be spent answering support calls. Documentation is part of the overall user experience, and is one of the biggest factors for increased API growth and usage.

## Challenges of API documentation

But documentation can be a tedious process. Two of the biggest documentation issues that teams can run into include: maintenance and interaction between multiple web services.

APIs, like so many other products, tend to evolve rapidly, during the development, and in some cases, in the release cycle. Maintaining and updating this documentation for your development team and end consumers, so they work with the API efficiently, becomes a difficult process. This is especially true if you're using static documents, like a .pdf, to provide documentation to your end consumers.

The second issue is facilitating interaction between multiple web services. Applications are made up of multiple services that constantly communicate and interact with each other. As RESTful services grow in number, so do the programming languages that are used to implement them, making it harder for them to communicate. API Documentation can be thought of as the interface for consuming an API, and such, could have had to facilitate this interaction between these different web services. Regular API interfaces, be it text documentation, or others like Javadocs, do not allow them to communicate with each other.

These challenges, along with other API painpoints, led to the creation of Swagger, a framework for describing the operations of a RESTful API. In the next section, we'll take a closer look at how Swagger can help address your documentation challenges.

## Why Use Swagger?

Swagger is an open source, human and machine readable API framework to design, build, and document APIs. What this means is that Swagger defines an API's RESTful contract, allowing all the API's stakeholders, be it your development team, or your end consumers, to understand what the API does and interact with its various resources, without having to integrate it into their own application. This contract is language agnostic and human readable, allowing both machines and humans to parse and understand what the API is supposed to do.

The Swagger specification can be written in YAML and JSON. Here's an example of what the API contract would look like:

### The Swagger specification, defined in YAML


```
swagger: '2.0'
info:
  version: 1.0.0
  title: Echo
  description: |
    ##### Echos back every URL, method, pa-
parameter and header
    Feel free to make a path or an opera-
tion and use **Try
Operation** to test it. The echo server
will
    render back everything.
schemes:
  - http
host: mazimi-prod.apigee.net
basePath: /echo
paths:
  /echo:
    get:
      responses:
        200:
          description: Echo GET
    post:
      responses:
        200:
          description: Echo POST
      parameters:
        - name: name
          in: formData
          description: name
          type: string
        - name: year
          in: formData
          description: year
          type: string
```

### The Swagger specification, defined in JSON

```
{
  "swagger": "2.0",
  "info": {
    "version": "1.0.0",
    "title": "Echo",
    "description": "##### Echos back every
URL, method, parameter and header\nFeel
free to make a path or an operation and
use **Try Operation** to test it. The echo
server will\nrender back everything.\n"
  },
  "schemes": [
    "http"
  ],
  "host": "mazimi-prod.apigee.net",
  "basePath": "/echo",
  "paths": {
    "/echo": {
      "get": {
        "responses": {
          "200": {
            "description": "Echo GET"
          }
        }
      },
      "post": {
        "responses": {
          "200": {
            "description": "Echo POST"
          }
        },
        "parameters": [
          {
            "name": "name",
            "in": "formData",
            "description": "name",
            "type": "string"
          },
          {
            "name": "year",
            "in": "formData",
            "description": "year",
```

You can see in the above example how the Swagger contract describes what the API does, it's request parameters and response objects, all without any indication of code implementation.

Web services defined with Swagger can communicate with each other irrespective of the language they're built in, since Swagger is language agnostic and machine readable. This contract can be converted to beautiful, interactive documentation for the API's end consumers using the Swagger UI automatically, so the issue of maintaining and updating documentation is minimal. The interactive documentation for the above contract would look as shown below:


**swagger**

## Echo

Echos back every URL, method, parameter and header  
 Feel free to make a path or an operation and use **Try Operation** to test it. The echo server will render back everything.

**default** Show/Hide List Operations Expand Operations

GET /echo

### Response Messages

HTTP Status Code	Reason	Response Model	Headers
200	Echo GET		

Try it out!

POST /echo

### Parameters

Parameter	Value	Description	Parameter Type	Data Type
name	<input type="text"/>	name	formData	string
year	<input type="text"/>	year	formData	string

### Response Messages

HTTP Status Code	Reason	Response Model	Headers
200	Echo POST		

Try it out!

Thus, the Swagger framework helps define a language agnostic, human readable format for APIs, that eases implementation, drives adoption, and stabilizes development. Since its creation, Swagger has become the world's most popular framework for API development, with over one million monthly downloads of the open source Swagger tools, and has evolved into the industry standard for designing and documenting RESTful APIs.

# Getting Some Swagger for Your APIs

There are obvious benefits of generating a Swagger definition for your API. The question then arises: How do we generate a Swagger specification for an existing API?

There are two general processes followed for generating a Swagger specification for an API. We've broadly identified them based on how the generation occurs. Keep in mind that Swagger is supported by an expansive community of open source and commercial tools. We'll introduce just a few of the tools you can use to generate a Swagger file.

## Swagger Generation During Runtime

In this method, the Swagger contract is generated from an API based on the meta-data added against the various resources, methods and controllers. This meta-data will generate the Swagger contract, client-side code, and other artifacts during runtime. Typically, this meta-data would be in the form of code annotations. The tools trigger as the various methods and functions are called against their resources, and produces the Swagger contract from the metadata defined in the API.

[Swagger Core](#) for [JAX-RS](#)-based APIs is one such example of a tool that generates the Swagger definition on runtime. To better elaborate this process, let's consider a case where we have to generate the Swagger specification from an API coded using JAX-RS, with the Jersey framework.

**There are three steps required to generate a Swagger document from an existing API:**

1. Adding dependencies to your application
2. Hooking Swagger Core to the application
3. Initialize the Swagger contract

The Swagger project uses maven for build and deployment of artifacts, available on Maven Central. These maven dependencies would need to be added to your JAX-RS coded API for Swagger Core to run.

A typical maven dependency would look like the one shown below:

```
<dependency>
  <groupId>io.swagger</groupId>
  <artifactId>swagger-jersey-jaxrs</artifactId>
  <version>1.5.12</version>
</dependency>
```

Because of differences in major versions of the Jersey REST framework, users should use the `swagger-jersey2-jaxrs` dependency for Jersey 2.x. The next step is to hook up Swagger Core into your API. Depending on the way Jersey is configured in your web service, you could hook up Swagger Core to your application using Spring, the Jersey's container Servlet, or manually.

Finally, based on the code annotations added in the previous steps, the Swagger definition can be initialized within your application during its runtime. The generated Swagger definition will be in two files, defined in JSON and YAML respectively.

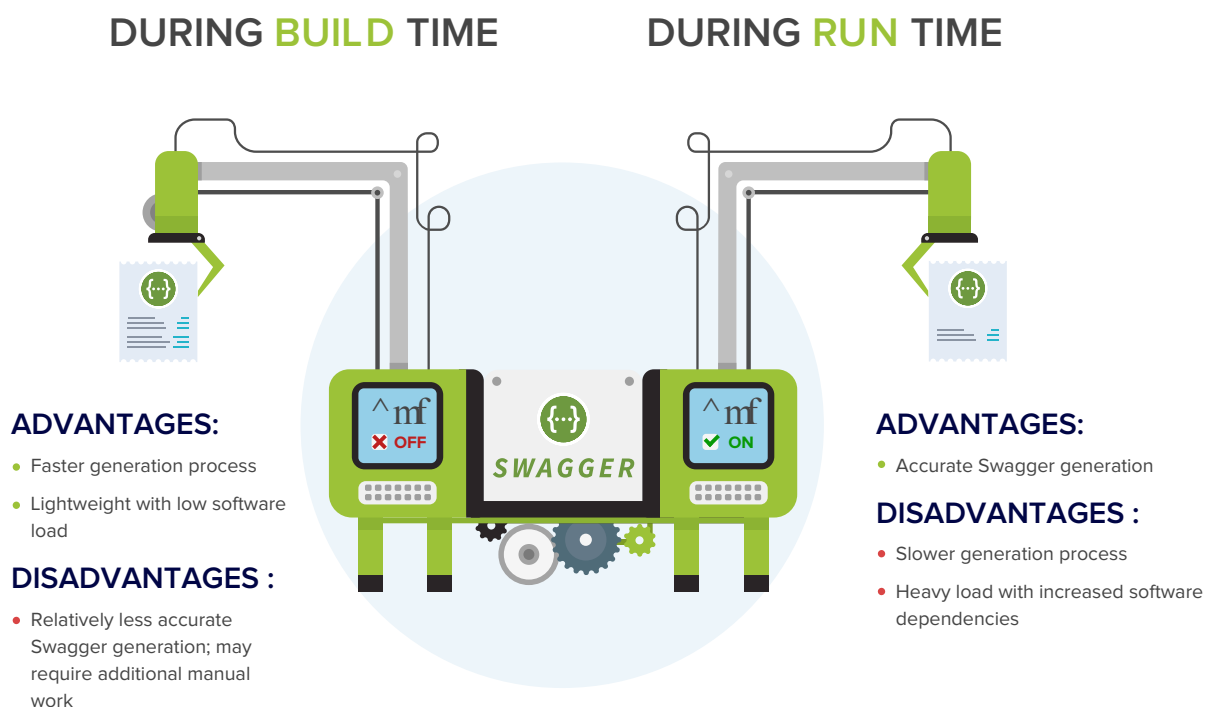
**Here are some additional resources to better understand this process:**

1. [Generating Swagger for a Jersey Project](#)
2. [Generating Swagger for Spring based APIs](#)
3. [Generating Swagger for PHP based APIs](#)

## Swagger Generation During Build Time

In this method, the Swagger contract is generated when preprocessing the API, that is, before runtime. Comments against various resources, methods and functions within the API help generate the Swagger specification. These comments are usually in a predefined, special syntax, based on the type of tool you use to generate the contract. The tool scans your API code for these special comments and produces the Swagger contract as an output. [Cakephp-swagger](#) and [grape-swagger](#) are prominent examples of tools that generate the Swagger contract during build time.

There are disadvantages and advantages offered by both methods. Generating the Swagger specification during runtime produces a more accurate API contract from the code, at the cost of software load in the form of additional dependencies, development time, and may add some overhead to the system. Conversely, generating the Swagger contract before runtime of the API is a more lightweight process, but there's a good chance that the Swagger contract produced may not accurately describe your API, as it must be manually maintained. In both approaches, there will likely be some additional work needed to ensure the Swagger file generated accurately represents the operations of your API.





# Documenting the API from the Swagger Specification

The generated specification is the basis of your API's technical and interactive documentation. A typical example of the generated JSON will be something like this:

```

"swagger": "2.0", "info": {"description": "This is a sample server Petstore server. You can find out more at Swagger at http://swagger.io or on irc.freenode.net, #swagger [http://swagger.io/irc/]. For this sample, you can use the api key 'special-key' to test the authorization filters.", "version": "1.0.0", "title": "Swagger Petstore", "termsOfService": "http://swagger.io/terms/", "contact": {"email": "apiteam@swagger.io"}, "license": {"name": "Apache 2.0", "url": "http://www.apache.org/licenses/LICENSE-2.0.html"}}, "schemes": ["http"], "securityDefinitions": {"api_key": {"type": "apiKey", "name": "special-key", "in": "header"}}, "paths": {""/": {"get": {"summary": "Find out more about our store", "url": "http://swagger.io/"}}, "/pets": {"post": {"tags": ["pet"], "summary": "Add a new pet to the store", "description": "", "operationId": "addPet", "consumes": ["application/json", "application/xml"], "produces": ["application/xml", "application/json"], "parameters": [{"in": "body", "name": "body", "description": "Pet object that needs to be added to the store", "required": true, "schema": {"$ref": "#/definitions/pet"}}, {"in": "header", "name": "api_key", "required": false, "type": "string", "description": "API key that must be presented to the server"}], "responses": {"400": {"description": "Invalid input data"}, "404": {"description": "Pet not found"}, "200": {"description": "Pet successfully created", "schema": {"$ref": "#/definitions/pet"}}}}, "put": {"summary": "Update an existing pet", "description": "", "operationId": "updatePet", "consumes": ["application/json", "application/xml"], "produces": ["application/xml", "application/json"], "parameters": [{"in": "body", "name": "body", "description": "Pet object that needs to be added to the store", "required": true, "schema": {"$ref": "#/definitions/pet"}}, {"in": "header", "name": "api_key", "required": false, "type": "string", "description": "API key that must be presented to the server"}], "responses": {"400": {"description": "Invalid ID supplied"}, "404": {"description": "Pet not found"}, "405": {"description": "Validation exception"}, "security": [{"petstore_auth": ["write:pets", "read:pets"]}]}}, "delete": {"summary": "Delete a pet", "description": "", "operationId": "deletePet", "consumes": ["application/xml", "application/json"], "produces": ["application/xml", "application/json"], "parameters": [{"in": "body", "name": "body", "description": "Pet object that needs to be added to the store", "required": true, "schema": {"$ref": "#/definitions/pet"}}, {"in": "header", "name": "api_key", "required": false, "type": "string", "description": "API key that must be presented to the server"}], "responses": {"400": {"description": "Invalid ID supplied"}, "404": {"description": "Pet not found"}, "405": {"description": "Validation exception"}, "security": [{"petstore_auth": ["write:pets", "read:pets"]}]}}, "writePets": {"get": {"tags": ["pet"], "summary": "Finds Pets by status", "description": "Multiple tags may be specified as comma separated strings, e.g. tag1,tag2.", "url": "http://swagger.io/pets/findByStatus", "consumes": ["application/xml", "application/json"], "produces": ["application/xml", "application/json"], "parameters": [{"in": "query", "name": "status", "description": "Status value that needs to be considered for filtering", "required": true, "type": "array", "items": {"type": "string", "enum": ["available", "pending", "sold"], "default": "available"}, "collectionFormat": "multi"}, {"in": "header", "name": "api_key", "required": false, "type": "string", "description": "API key that must be presented to the server"}], "responses": {"200": {"description": "successful operation", "schema": {"$ref": "#/definitions/pet"}}, "400": {"description": "Invalid status value"}, "405": {"description": "Validation exception"}, "security": [{"petstore_auth": ["write:pets", "read:pets"]}]}}, "writePetsTags": {"get": {"tags": ["pet"], "summary": "Finds Pets by tags", "description": "Multiple tags may be specified as comma separated strings, e.g. tag1,tag2.", "url": "http://swagger.io/pets/findByTags", "consumes": ["application/xml", "application/json"], "produces": ["application/xml", "application/json"], "parameters": [{"in": "query", "name": "tags", "description": "Tags to filter by", "required": true, "type": "array", "collectionFormat": "multi"}, {"in": "header", "name": "api_key", "required": false, "type": "string", "description": "API key that must be presented to the server"}], "responses": {"200": {"description": "successful operation", "schema": {"$ref": "#/definitions/pet"}}, "400": {"description": "Invalid tag value"}, "405": {"description": "Validation exception"}, "security": [{"petstore_auth": ["write:pets", "read:pets"]}]}}, "petId": {"get": {"tags": ["pet"], "summary": "Find pet by ID", "description": "Returns a single pet", "operationId": "getPetById", "produces": ["application/xml", "application/json"], "parameters": [{"in": "path", "name": "petId", "description": "ID of pet to return", "required": true, "type": "integer", "format": "int64"}, {"in": "header", "name": "api_key", "required": false, "type": "string", "description": "API key that must be presented to the server"}], "responses": {"400": {"description": "Invalid ID supplied"}, "404": {"description": "Pet not found"}, "200": {"description": "successful operation", "schema": {"$ref": "#/definitions/pet"}}}}, "petId": {"put": {"tags": ["pet"], "summary": "Update status of the pet", "description": "Updates status of the pet", "required": false, "type": "string", "enum": ["available", "pending", "sold"], "default": "available"}, {"in": "header", "name": "api_key", "required": false, "type": "string", "description": "API key that must be presented to the server"}], "responses": {"400": {"description": "Invalid ID supplied"}, "404": {"description": "Pet not found"}, "200": {"description": "successful operation", "schema": {"$ref": "#/definitions/pet"}}}}, "petId": {"delete": {"tags": ["pet"], "summary": "Delete a pet", "description": "Deletes a pet", "required": false, "type": "string", "enum": ["available", "pending", "sold"], "default": "available"}, {"in": "header", "name": "api_key", "required": false, "type": "string", "description": "API key that must be presented to the server"}], "responses": {"400": {"description": "Invalid ID supplied"}, "404": {"description": "Pet not found"}, "200": {"description": "successful operation", "schema": {"$ref": "#/definitions/pet"}}}}, "uploadImage": {"post": {"tags": ["pet"], "summary": "Uploads an image", "description": "Uploads an image", "operationId": "uploadImage", "consumes": ["multipart/form-data"], "produces": ["application/xml", "application/json"], "parameters": [{"in": "body", "name": "image", "description": "Image to upload", "required": true, "type": "file"}, {"in": "header", "name": "api_key", "required": false, "type": "string", "description": "API key that must be presented to the server"}], "responses": {"200": {"description": "successful operation", "schema": {"$ref": "#/definitions/petResponse"}}, "400": {"description": "Invalid ID supplied"}, "404": {"description": "Pet not found"}, "200": {"description": "successful operation", "schema": {"$ref": "#/definitions/petResponse"}}, "security": [{"petstore_auth": ["write:pets", "read:pets"]}]}}, "storeInventory": {"get": {"tags": ["store"], "summary": "Returns pet inventories by status", "description": "Returns a map of status codes to inventories", "operationId": "getInventory", "consumes": ["application/xml", "application/json"], "produces": ["application/xml", "application/json"], "parameters": [{"in": "header", "name": "api_key", "required": false, "type": "string", "description": "API key that must be presented to the server"}], "responses": {"200": {"description": "successful operation", "schema": {"$ref": "#/definitions/inventory"}}, "400": {"description": "Invalid ID supplied"}, "404": {"description": "Pet not found"}, "200": {"description": "successful operation", "schema": {"$ref": "#/definitions/inventory"}}, "security": [{"petstore_auth": ["write:pets", "read:pets"]}]}}, "order": {"post": {"tags": ["order"], "summary": "Place an order for a pet", "description": "", "operationId": "placeOrder", "produces": ["application/xml", "application/json"], "parameters": [{"in": "body", "name": "body", "description": "order placed for purchasing the pet", "required": true, "schema": {"$ref": "#/definitions/order"}}, {"in": "header", "name": "api_key", "required": false, "type": "string", "description": "API key that must be presented to the server"}], "responses": {"200": {"description": "successful operation", "schema": {"$ref": "#/definitions/order"}}, "400": {"description": "Invalid Order"}, "404": {"description": "Store/order/orderId"}, {"get": {"tags": ["store"], "summary": "Find purchase order by ID", "description": "Find purchase order by ID, will return order when user values all generated extensions", "operationId": "getOrderById", "produces": ["application/xml", "application/json"], "parameters": [{"in": "path", "name": "orderId", "description": "ID of pet that needs to be fetched", "required": true, "type": "integer", "maximum": 10.0, "minimum": 1.0, "format": "int64"}, {"in": "header", "name": "api_key", "required": false, "type": "string", "description": "API key that must be presented to the server"}], "responses": {"200": {"description": "successful operation", "schema": {"$ref": "#/definitions/order"}}, "400": {"description": "Invalid ID supplied"}, "404": {"description": "Order not found"}, "delete": {"tags": ["store"], "summary": "Delete purchase order by ID", "description": "For valid response try integer IDs with positive suffix to generate IDs, for example \"1011\" as \"1011delete\". If the order has been deleted, \"404\" will return.", "operationId": "deleteOrder", "consumes": ["application/xml", "application/json"], "parameters": [{"in": "path", "name": "orderId", "description": "ID of the order that needs to be deleted", "required": true, "type": "integer", "minimum": 1.0, "format": "int64"}, {"in": "header", "name": "api_key", "required": false, "type": "string", "description": "API key that must be presented to the server"}], "responses": {"400": {"description": "Invalid ID supplied"}, "404": {"description": "Order not found"}, "200": {"description": "successful operation", "schema": {"$ref": "#/definitions/order"}}, "security": [{"petstore_auth": ["write:pets", "read:pets"]}]}}, "user": {"get": {"tags": ["user"], "summary": "Create user", "description": "This can only be done by the logged in user.", "operationId": "createUser", "produces": ["application/xml", "application/json"], "parameters": [{"in": "body", "name": "body", "description": "Created user object", "required": true, "schema": {"$ref": "#/definitions/user"}}, {"in": "header", "name": "api_key", "required": false, "type": "string", "description": "API key that must be presented to the server"}], "responses": {"200": {"description": "successful operation", "schema": {"$ref": "#/definitions/user"}}, "400": {"description": "Invalid user data"}, "409": {"description": "User already exists"}, "security": [{"petstore_auth": ["write:pets", "read:pets"]}]}}, "user": {"post": {"tags": ["user"], "summary": "Create list of users with given input", "description": "Creates list of users with given input array", "operationId": "createUsersWithArrayInput", "produces": ["application/xml", "application/json"], "parameters": [{"in": "body", "name": "body", "description": "List of user object", "required": true, "schema": {"type": "array", "items": {"$ref": "#/definitions/user"}}, {"in": "header", "name": "api_key", "required": false, "type": "string", "description": "API key that must be presented to the server"}], "responses": {"200": {"description": "successful operation", "schema": {"$ref": "#/definitions/user"}}, "400": {"description": "Invalid user data"}, "409": {"description": "User already exists"}, "security": [{"petstore_auth": ["write:pets", "read:pets"]}]}}, "user": {"put": {"tags": ["user"], "summary": "Create list of users with given input", "description": "Creates list of users with given input array", "operationId": "createUsersWithInput", "produces": ["application/xml", "application/json"], "parameters": [{"in": "body", "name": "body", "description": "List of user object", "required": true, "schema": {"type": "array", "items": {"$ref": "#/definitions/user"}}, {"in": "header", "name": "api_key", "required": false, "type": "string", "description": "API key that must be presented to the server"}], "responses": {"200": {"description": "successful operation", "schema": {"$ref": "#/definitions/user"}}, "400": {"description": "Invalid user data"}, "409": {"description": "User already exists"}, "security": [{"petstore_auth": ["write:pets", "read:pets"]}]}}, "user": {"get": {"tags": ["user"], "summary": "Logs user into the system", "description": "", "operationId": "loginUser", "produces": ["application/xml", "application/json"], "parameters": [{"in": "body", "name": "body", "description": "Log user into the system", "required": true, "schema": {"$ref": "#/definitions/login"}}, {"in": "header", "name": "api_key", "required": false, "type": "string", "description": "API key that must be presented to the server"}], "responses": {"200": {"description": "successful operation", "schema": {"$ref": "#/definitions/token"}}, "400": {"description": "Invalid username/password"}, "401": {"description": "Unauthorized"}, "404": {"description": "User not found"}, "200": {"description": "successful operation", "schema": {"$ref": "#/definitions/token"}}, "security": [{"petstore_auth": ["write:pets", "read:pets"]}]}}, "logout": {"get": {"tags": ["user"], "summary": "Logs out current logged in user session", "description": "", "operationId": "logoutUser", "produces": ["application/xml", "application/json"], "parameters": [{"in": "body", "name": "body", "description": "Log out current logged in user session", "required": true, "schema": {"$ref": "#/definitions/logout"}}, {"in": "header", "name": "api_key", "required": false, "type": "string", "description": "API key that must be presented to the server"}], "responses": {"200": {"description": "successful operation", "schema": {"$ref": "#/definitions/logout"}}, "401": {"description": "Unauthorized"}, "200": {"description": "successful operation", "schema": {"$ref": "#/definitions/logout"}}, "security": [{"petstore_auth": ["write:pets", "read:pets"]}]}}, "petStore": {"get": {"tags": ["petstore"], "summary": "Returns all pets from the system", "description":
```

This contract is in no way the final, technical documentation. This generated specification would have to be further expanded by tech writers to be called actual documentation. Documentation from the generated contract would mean adding meaningful, understandable information that your end consumers can use to achieve API success. This information would be added in the metadata of the API contract.

Documentation can be a tricky process. It's a manual, collaborative operation that expects a lot of time, quality and empathy from the writers. When traversing the journey from API code to documentation, the most important thing to have is a seamless workflow that doesn't make you break a sweat with additional setup. It is usually recommended to give API documentation its own, unique care and treatment, since documentation is the first interface that's used by users and customers to consume your API offering.

[SwaggerHub](#), a platform for developing and documenting APIs using Swagger, is one such tool for writing comprehensive documentation. It is a dedicated platform for documentation, with all the configuration and hosting taken care of, allowing you to seamlessly integrate documentation into your API workflow.

This is supported by additional features like centralized cloud storage, forking, merging, pushing to source control hosts and deploying to API management platforms, so your entire API lifecycle can be orchestrated from one single focal point.



Once your API's contract is generated from your existing API code, you can import it in SwaggerHub, and continue your API journey. Once the specification is imported, the interactive documentation is automatically generated and hosted on SwaggerHub. The specification can be edited, with your technical writers adding the right information in your API that can gives its consumers the required information to integrate with it.



SwaggerHub's built-in tools further assist in the documentation process. Some of them include:

- **Compare & Merge:** Compare and merge your API documentation stored in SwaggerHub with an other API generated from your integration process, be it in your local machine or to another definition stored in SwaggerHub. [Learn more.](#)
- **Collaboration & Issue Tracking:** Work with multiple stakeholders on a centralized platform. SwaggerHub provides the platform for teams to collaborate throughout the design and documentation process, collecting feedback and tracking issues in real-time with comments in the SwaggerHub editor. [Learn more.](#)
- **Secure Cloud-Hosting:** Store your API definitions and documentation in a platform built for APIs. SwaggerHub autosaves your work throughout the documentation process and provides a central place to host your documentation, without the need to setup a server. [Learn more.](#)

Adding human readable documentation to your API's contract is what an API should aspire for, after which the documentation can be hosted in the environment of your choice. Once the documentation is done, users can publish the API for consumption. either on SwaggerHub, or directly to API Management platforms like AWS API Gateway or Microsoft Azure API Management systems.

## Closing Notes

Documentation is the usage manual of your API, and is one of the biggest drivers to achieving your API's business goals. The Swagger framework alleviates documentation concerns, creating interactive, human and machine readable documentation, that's auto generated and needs minimal maintenance. Generating the Swagger definition from an API's code could mean more work, but the contract opens the doors to the good, useful API documentation that begets better adoption and consumption, which is what every API should strive.

## Additional Resources

Hopefully, this whitepaper provided the guidance your team needs to start documenting your existing APIs with Swagger. In addition to the approaches outlined in this whitepaper, a growing number of organizations are also adopting a “design-first” approach to Swagger, which starts with writing your Swagger definition and using that to drive the development of your API.

SwaggerHub’s integrated development platform supports both approaches to documenting your APIs with Swagger. Below is a list of resources your team can use to implement the steps outlined in this whitepaper. When you’re ready you can sign up to [try SwaggerHub for free](#). We offer a free version of the tool, which individuals developers and designers can use to start documenting their APIs with Swagger. We also offer professional plans, with advanced capabilities to enable your team to collaborate on your API’s documentation and control who can view and access the API. We also offer integrations to tools like source control, API management platforms, testing solutions, and more.

### SwaggerHub

SwaggerHub is an integrated API development platform built for teams, that combines the capabilities of the open source Swagger tools, namely the Swagger Editor, Swagger UI and Swagger Codegen, with advanced and unique features to build, document and deploy APIs. SwaggerHub enables development teams to collaborate and coordinate the entire lifecycle of an API with the flexibility to integrate with the toolset of your choice.

Learn More: [SwaggerHub.com](https://swaggerhub.com)

### Swagger Open Source

Swagger is an open source API framework, sponsored by SmartBear Software, that allows developers and teams to design, build, document and consume RESTful web services. The Swagger framework drives consistency and stability across the API workflow in a way that’s both machine and human readable. With over 10 million downloads and the industry standard for describing REST APIs, Swagger is the world’s most popular framework for RESTful services.

Learn more: [Swagger.io](https://swagger.io)

### Swagger Community

The Swagger ecosystem is spearheaded by one of the most active developer communities in open source. You can find a few resources to connect with the community and get started with the tools or contribute to the development of the ecosystem.

Learn more: [Swagger.io/Community](https://swagger.io/Community)



***TRY SWAGGERHUB FOR FREE***  
*SWAGGERHUB.COM*