Swagger 3.0

Basic Structure

You can write OpenAPI definitions in YAML or JSON. A sample OpenAPI 3.0 definition written in YAML looks like:

```yaml
openapi: 3.0.0
info:
  title: Sample API
  description: Optional multiline or single-line description in [CommonMark](http://commonma
  version: 0.1.9

servers:
  - url: http://api.example.com/v1
    description: Optional server description, e.g. Main (production) server
  - url: http://staging-api.example.com
    description: Optional server description, e.g. Internal staging server for testing

paths:
  /users:
    get:
      summary: Returns a list of users.
      description: Optional extended description in CommonMark or HTML.
      responses:
        '200':    # status code
          description: A JSON array of user names
          content:
            application/json:
              schema:
                type: array
                items:
                  type: string
```

Fig 1: Sample OpenAPI 3.0 Definition

Metadata

Every API definition must include the version of the OpenAPI Specification that this definition is based on. The OpenAPI version defines the overall structure of an API definition – what you can document and how you document it. OpenAPI 3.0 uses semantic versioning with a three-part version number. The available versions are 3.0.0, 3.0.1, 3.0.2, and 3.0.3; they are functionally the same. The info section contains API information: title, description (optional), version.

Servers

The servers section specifies the API server and base URL. You can define one or several servers. All API paths are relative to the server URL.

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:

```yaml
paths:
  /users:
    get:
      summary: Returns a list of users.
      description: Optional extended description in CommonMark or HTML
      responses:
        '200':
          description: A JSON array of user names
          content:
            application/json:
              schema:
                type: array
                items:
                  type: string
```

Fig 2: Path example

Parameters

Operations can have parameters passed via URL path (/users/{userId}), query string (/users?role=admin), headers (X-CustomHeader: Value) or cookies (Cookie: debug=0). You can define the parameter data types, format, whether they are required or optional, and other details.

Request Body

Request bodies are typically used with “create” and “update” operations (POST, PUT, PATCH). For example, when creating a resource using POST or PUT, the request body usually contains the representation of the resource to be created. OpenAPI 3.0 provides the requestBody keyword to describe request bodies.

Responses

An API specification needs to specify the responses for all API operations. Each operation must have at least one response defined, usually a successful response. A response is defined by its HTTP status code and the data returned in the response body and/or headers.

Inputs and Output Models

The global components/schemas section lets you define common data structures used in your API. They can be referenced via $ref whenever a schema is required – in parameters, request bodies, and response bodies.

```json
{
  "id": 4,
  "name": "Arthur Dent"
}
```

Fig 3: A JSON Object

The JSON object in fig 3 can represented as:

```yaml
components:
  schemas:
    User:
      type: object
      properties:
        id:
          type: integer
          example: 4
        name:
          type: string
          example: Arthur Dent
      # Both properties are required
      required:
        - id
        - name
```

Fig 4: Defining schema

And then referenced in the request body schema and response body schema as follows:

```yaml
paths:
  /users/{userId}:
    get:
      summary: Returns a user by ID.
      parameters:
        - in: path
          name: userId
          required: true
          schema:
            type: integer
            format: int64
            minimum: 1
      responses:
        '200':
          description: OK
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/User'    # <-------
  /users:
    post:
      summary: Creates a new user.
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/User'      # <-------
      responses:
        '201':
          description: Created
```

Fig 5: Referencing the schema

Authentication

OpenAPI uses the term **security scheme** for authentication and authorization schemes. OpenAPI 3.0 lets you describe APIs protected using the following security schemes:

- HTTP authentication schemes (they use the Authorization header):
    - Basic
    - Bearer
    - other HTTP schemes as defined by RFC 7235 and HTTP Authentication Scheme Registry
- API keys in headers, query string or cookies
    - Cookie authentication
- OAuth 2
- OpenID Connect Discovery