Proposed OpenAPI 2.0 / Swagger Style Guide

Info Section

API Title

The API title specified by the info.title field MUST be present and a non-empty string.

The API title should be descriptive and reflect the purpose of the API.

API Description

The info.description field provides a short overview of the API and MUST be present and a non-empty string.

The API Description should be used to provide the purpose of the API as well as how and why it should be used.

The API Description may include information about basic instructions (e.g., Getting Started) to users on how to use the API, such as host port, authentication, common error codes, test environment, etc., as well as links to additional resources (such as ONAP Use Cases), plus other details that would help other developers start using the API.

The API Description should provide some basic instructions to users on how to use a Swagger viewer (e.g., Redoc, Swagger UI, etc.).

The API Description may be multiline, and GitHub Flavored Markdown, GFM syntax, can be used for rich text representation.

API Contact

The info.contact section provides contact information about the API, including: name, url, and email address.

In ONAP, the info.contact.name field provides the identifying name of the contact organization, and MUST be set to "ONAP"

In ONAP, the info.contact.url field MUST be set to "https://onap.readthedocs.io"

In ONAP, the info.contact.email field MUST be set to "onap-discuss@lists.onap.org"

API License

The info.license section specifies the license name and url for the API.

In ONAP, the info.license.name field MUST be set to "Apache 2.0"

In ONAP, the info.license.url field MUST be set to "http://www.apache.org/licenses/LICENSE-2.0"

API Version

The info.version field is used to provide the version of the application API.

Note: Align URLs to include only the MAJOR version. For example, the base path MUST only contain the MAJOR version number.

In ONAP, the info.version field MUST be set to the fully-qualified version number of the Swagger file (ex: 1.4.18). Further information about versioning is ONAP may be found at: https://wiki.onap.org/display/DW/ONAP+API+Common+Versioning+Strategy+%28CVS%29+Guidelines

Extension Fields

Within the info section, the following are extensions of the Swagger specification and SHALL be required for ONAP:

The x-planned-retirement-date field MUST use YYYYMM; string type. This is the date that the API shall be deprecated, based on the BWC Policy. NOTE: APIs may be active after their retirement date, but are not guaranteed to remain in production. An API retirement may be pushed out to accommodate backwards compatibility for clients.

The x-component field is a string type filed that describes the ONAP component that primarily owns the API from a development perspective. For examples, SDC, MSO, SNIRO, etc., or the mS name.

Host

The host field must specify the host name serving the API. This MUST be the host only and does not include the scheme nor sub-paths. It MAY include a port. (e.g., serverRoot:54321)

Base Path

The *basePath* field describes base path, relative to the host, on which the API is served. The value MUST start with a leading slash (/). The base path MUST only contain the MAJOR version number to minimize changes in the URL for MINOR and PATCH releases.

For ONAP the *basePath* field MAY be in the form: "/"<*projectName*>"/"<*apiName*>"/v"<*majorVersionNumber*>

Path Extensions

Under the Path, the x-interface info extension shall be required. This extension contains two attributes:

api-version - fully-qualified version number of the API (ex: 1.3.6); string type. This is the version of the API. This differs from the version in in the Info filed. Components shall follow the Versioning Use Cases above to determine how to evolve API versions.

last-mod-release - use release number or name (this should be consistent, choose either one); string type. This is the last release that the API was modified in.

Path Operations

Operation Id

The operationId field within a path operation MUST be present and non-empty.

The operationId field within a path operation MUST be unique among all operations described in the API.

The operationId field within a path operation SHOULD follow common programming naming conventions.

For ONAP, it is recommended that the operationId field within a path operation be formed as: <objectName><operation>. (e.g., petsGet) using camelCase for word separation.

Operation Summary

The summary field within a path operation MUST be present and non-empty.

The summary field within a path operation should be short, descriptive and reflect the purpose of the operation. Please limit the summary to 5 to 10 words only, and no more than 120 characters.

Operation Description

The description field within a path operation MUST be present and non-empty.

The description field within a path operation should be used to provide the purpose of the operation as well as how and why it should be used, and document any prerequisites and usage specifics.

The description field within a path operation may be multiline, and GitHub Flavored Markdown, GFM syntax, can be used for rich text representation.

The description field within a path operation may provide example calls or usages of the operation.

Operation Tags

The tags field within a path operation MUST be present and non-empty.

Operation must have one and only one tag.

The tags field within a path operation is used to group operations logically into categories or topics, such as Account, Payments, Reports, Search, etc.

Operation Parameters

The description field within an operation parameter MUST be present and non-empty.

Operation Responses

Each Operation MUST have at least one successful (i.e. 2xx) response defined.

Each Operation should define a default response that covers responses not specifically defined for the Operation.

Models

Descriptions

Definition descriptions MUST be present and non-empty string for each model property.

Examples

All Model Properties that are not $refs should have an example value specified.

Naming Conventions

URL Construction

URI structure http://[host]:[port]/api/{service-name}]/v{version-number}/{resource}

The URI is comprised of a fixed base uri /api/{service-name}]/v{version-number} and the resource path. Only major version number is used in the URI. For example: http://127.0.0.1:8080/api/petstore/v1/pets/dogs

CRUD function names should not be used in URIs. Instead, CRUD actions should be represented by HTTP methods. Below is the proposed methodology to implement CRUD operations in a RESTful API.

A trailing forward slash (/) should not be included in URIs

Forward slash separator (/) must be used to indicate a hierarchical relationship

Prefer camelCase, but if necessary use Hyphens (-) instead of Underscores (_) if separation of words is needed in the URI

Lowercase letters should be preferred in URI paths

File extensions should not be included in URIs

A plural noun should be used for collection names

For example, the URI for a collection of dog documents uses the plural noun form of its contained resources: /api/petstore/v1/pets/dogs

Further Information may be found at: https://wiki.onap.org/display/DW/RESTful+API+Design+Specification

Pluralization of Resource names

Always use plurals in resource / node names to keep API URIs consistent across all operations.

Model Names

Model names should be simple, descriptive, and meaningful.

Model names should be in "upper camel case".

Property Names

Property names should be simple, meaningful, and descriptive with defined semantics.

Property names MUST be camelCased, ascii strings.

The first character MUST be a letter.

Subsequent characters can be a letter or a digit.

Reserved keywords should be avoided.

Array types should have plural property names. All other property names should be singular.

HTTP response status codes

The API specification should describe the right HTTP status code to return the client.

Status codes should align with IETF's HTTP Status Code Registry: https://www.ietf.org/assignments/http-status-codes/http-status-codes.xml

For example:

200 – OK – Everything is working

201 – OK – New resource has been created

204 – OK – The resource was successfully deleted

304 – Not Modified – The client can use cached data

400 – Bad Request – The request was invalid or cannot be served. The exact error should be explained in the error payload. E.g., "The JSON is not valid“

401 – Unauthorized – The request requires a user authentication

403 – Forbidden – The server understood the request but is refusing it or the access is not allowed.

404 – Not found – There is no resource behind the URI.

409 – Conflict – Whenever a resource conflict would be caused by fulfilling the request. E.g., Duplicate entries, deleting root objects when cascade-delete not supported, etc.

422 – Unprocessable Entity – Should be used if the server cannot process the entity, e.g. if an image cannot be formatted or mandatory fields are missing in the payload.

500 – Internal Server Error – API developers should avoid this error. If an error occurs in the global catch blog, the track trace should be logged and not returned as in the response.

All exceptions should be mapped in an error payload. Here is an example how a JSON payload may look:

```
{

  "message": "Sorry, the requested resource does not exist",

  "code": 34

}
```

Security