

Introduction to Open API Specification



Describing APIs

- Describe RESTful HTTP APIs in a machine-readable way
- API description is independent of outputs such as documentation
- Enable things that are not "just" documentation

More than docs, developers and docs don't
always ~~not~~ DevX: give them what they need to
succeed

Spec-First API Design



@lornajane

Cheaper to change a spec than rebuild a system. ^{if you can}
to quickly and with fewer vim macros

About OpenAPI Spec

API description language formerly known as "Swagger".

Became "OpenAPI Spec" -> v3 released

(some tools are still catching up on v3)

OAS will be to modern APIs what WSDL was for SOAP
some context on that

or to the open standard

New APIs or Existing Ones?

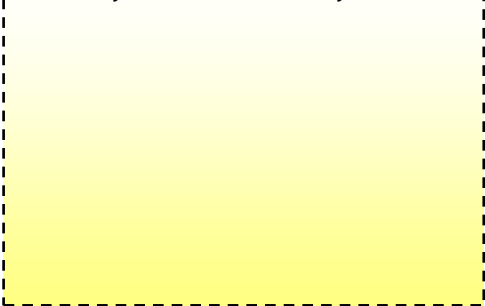
@lornajane

New APIs or Existing Ones?

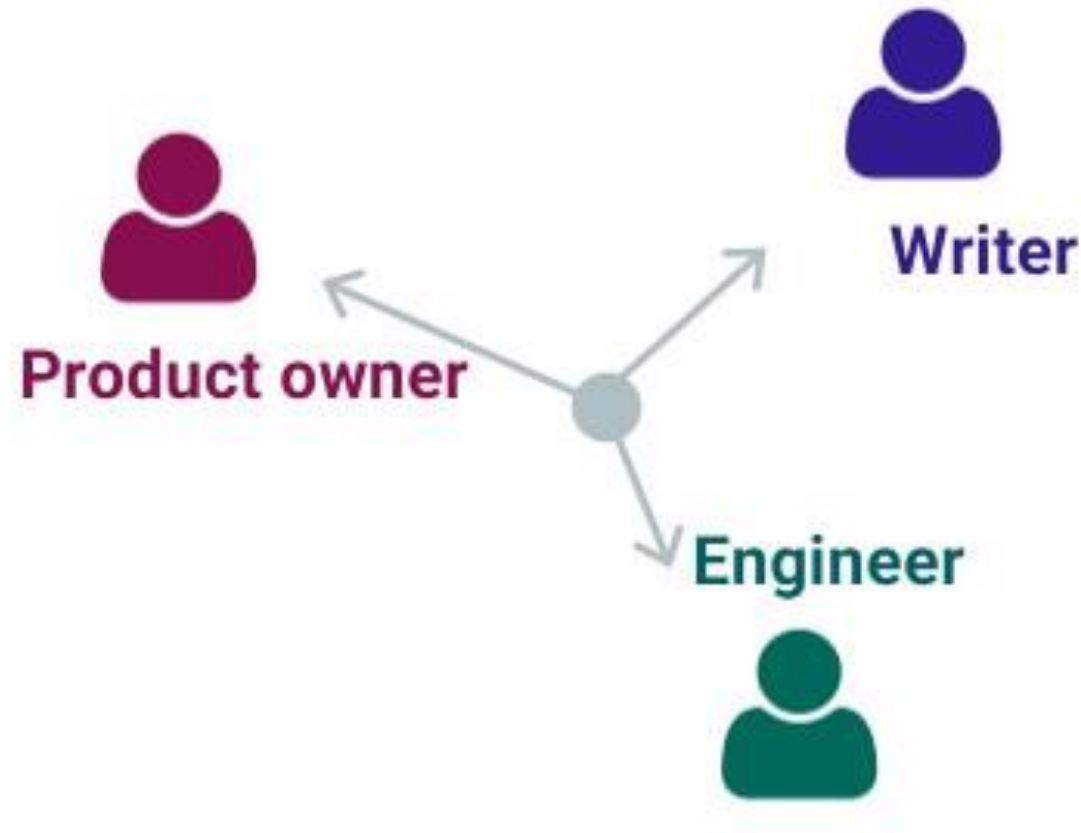
Yes!

@lornajane

It's not easy to retrofit but still totally worth it



Who Writes OpenAPI Specs?



I'm an Engineer with Writing tendencies
with ~~me~~ the others, this is right up your street

Anatomy of OpenAPI Spec



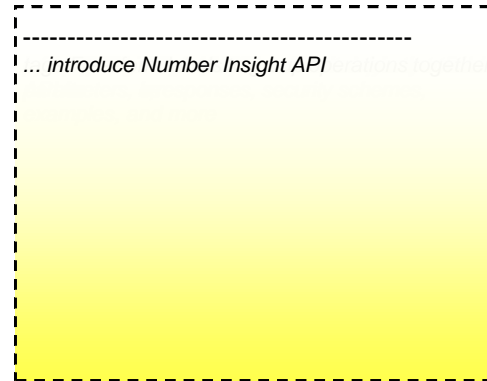
@lornajane

Spec is kind of a tree, fixed top level elements, and
things within each

Anatomy of OpenAPI Spec

Top-level elements:

- openapi
- info
- servers
- paths
- components
- security
- tags



OpenAPI Spec Examples

A JSON or YAML file holds the description (this is YAML)

```
openapi: 3.0.0
servers:
  - url: 'https://api.nexmo.com/ni'
info:
  title: Number Insight API
  version: 1.1.0
  description: >-
    Nexmo's Number Insight API delivers real-time intelligence about the val
... a few hundred more lines here
```

JSON or YAML? I could argue for either. YAML is ~~le~~the list you'd write in your text editor, JSON has more quotes and brackets

@lornajane

Documenting an Endpoint

```
paths:
  '/basic/{format}':
    parameters:
      - $ref: '#/components/parameters/format'
    get:
      operationId: getNumberInsightBasic
      parameters:
        - $ref: '#/components/parameters/number'
        - $ref: '#/components/parameters/country'
      responses:
        '200':
          description: OK
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/niResponseJsonBasic'
```

References are excellent - create reusable content ~~at~~ ^{rather than} copy/pasting (and the errors that go with that)

Example Parameter

```
number:
  name: number
  in: query
  description: 'A single phone number that you need insight about in national or
example: '447700900000'
  required: true
  schema:
    type: string
    pattern: '^[0-9-+\\(\\)\\s]*$'
```

Example Response

```
niResponseJsonBasic:
```

```
  type: object
```

```
  properties:
```

```
    status:
```

```
      $ref: '#/components/schemas/niBasicStatus'
```

```
    status_message:
```

```
      type: string
```

```
      description: 'The status description of your request.'
```

```
      example: 'Success'
```

```
    request_id:
```

```
      type: string
```

```
      description: 'The unique identifier for your request. This is a alphanumeric'
```

```
      example: 'aaaaaaaa-bbbb-cccc-dddd-0123456789ab'
```

```
      maxLength: 40
```

```
...
```

That looks complicated!



It's very verbose and a bit unwieldy, not really rocket



@lornajane

Rendered Example: ReDoc

Retrieve API Secrets

AUTHORIZATIONS:

basicAuth

PATH PARAMETERS

— account_id
required

string
Example: "abcd1234"
ID of the account

Responses

^ 200 API secret response

RESPONSE SCHEMA: application/json

— secrets >

object
A list of secrets under the "secrets" key.

v 401 Credential is missing or invalid

v 404 Resource could not be found

GET /accounts/{account_id}/secrets

Response samples

200 401 404

application/json

Copy Expand all Collapse all

```
{
  - "secrets": {
    - "secrets": [
      - {
        "id": "ad6dc56f-07b5-46e1-a527-85530e625800",
        "created_at": "2017-03-02T16:34:49Z"
      }
    ]
  }
}
```

It is standard! I use this while I'm spec-ing regardless of the eventual renderer

@lorna.jane

Rendered Example: Nexmo

Retrieve API Secrets

GET `https://api.nexmo.com/accounts/:account_id/secrets`

Authentication

Key	Description	Example	Default
Authorization	Base64 encoded API key and secret joined by a colon. Read more	Basic <base64>	None

Path Parameters

Key	Description	Example	Default
account_id <small>REQUIRED string</small>	ID of the account	abcd1234	None

[View response field descriptions](#)

HTTP response 200

```
{
  "secrets": {
    "secrets": [
      {
        "id": "ad6dc56f-07b5-46e1-a527-8553",
        "created_at": "2017-03-02T16:34:49Z"
      }
    ]
  }
}
```

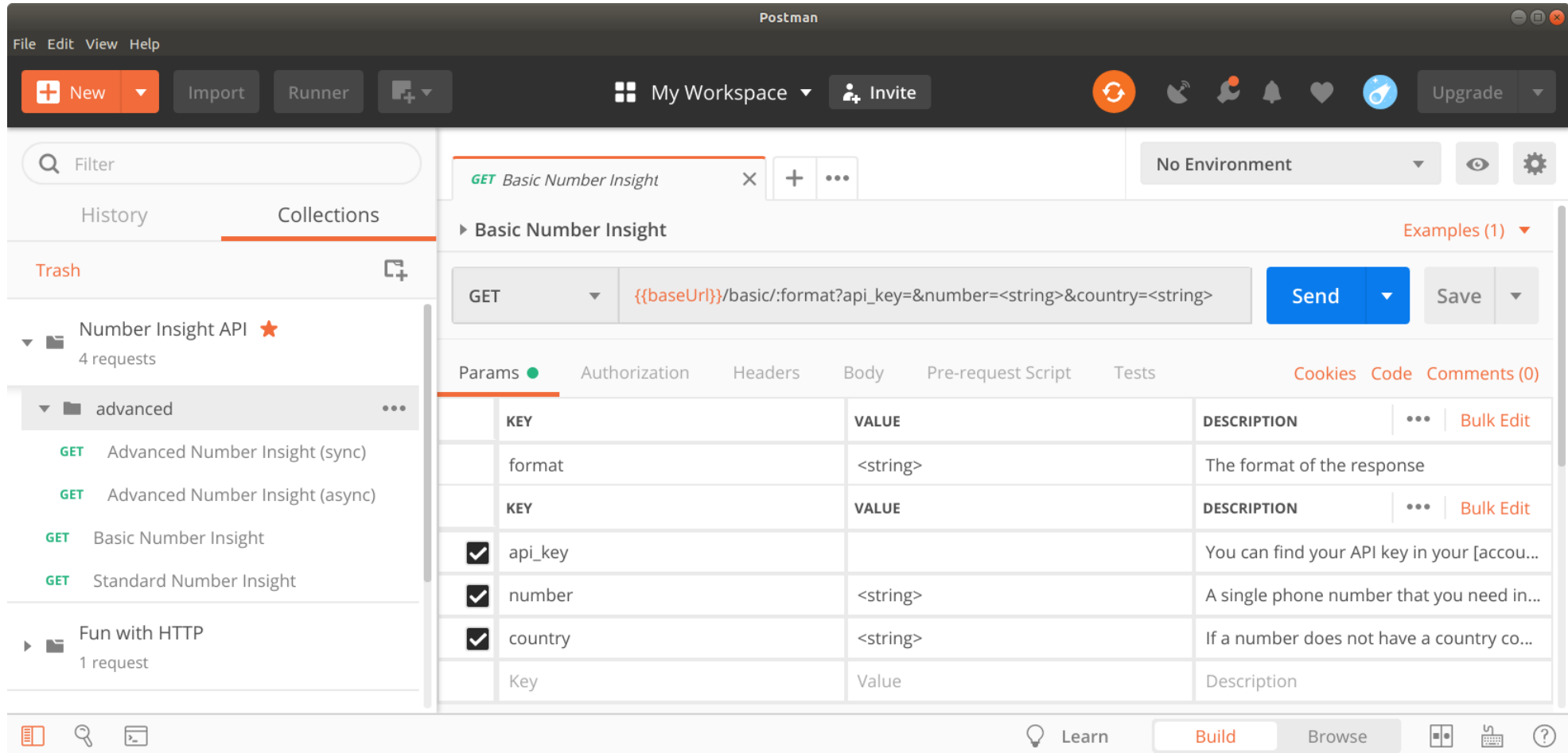
HTTP response 401

HTTP response 404

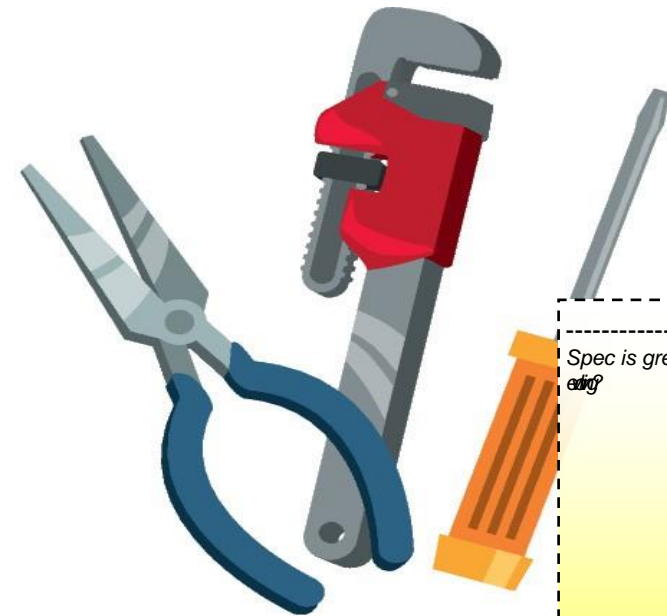
Rendered on Nexmo Developer Portal

@lornajane

Imported into Postman



Tools To Get Things Done



Spec is great, community is great. Tools are ~~also~~ ^{too}.

@lornajane

Please use Source Control



See also: <https://gitworkbook.com>

Editing Tools

There are editors with helpful tools

- I like Atom with `linter-swagger` <https://atom.io>
- Try SwaggerUI, SwaggerHub, etc <https://swagger.io/tools/>
- APICurio Studio gets good reviews <https://www.apicur.io/>
- Stoplight looks interesting <https://stoplight.io>

(feel free to tweet your best tools at me, I'll share them all lol)

I use vim but probably wouldn't preach that
not, also fine

@lornajane

OAS in Atom

The screenshot shows the Atom text editor with the file `secret-management.yml` open. The sidebar on the left displays the project structure, including a `.git` folder and the `secret-management.yml` file. The main editor area shows the following YAML content:

```
1 openapi: 3.0.0
2 servers:
3   - url: 'https://api.nexmo.com'
4 info:
5   version: 1.0.0
6   title: Secret Management API
7   description: >-
8     The Nexmo Secret Management API enables you to create one new API Secret in additi
9 security:
10  - basicAuth: []
11 paths:
12  /accounts/{account_id}/secrets:
13    get:
14      summary: Retrieve API Secrets
15      operationId: retrieveSecrets
16      parameters:
17        - $ref: '#/components/parameters/accountId'
18      responses:
19        '200':
20          description: API secret response
21          content:
```

The status bar at the bottom indicates the file is `secret-management.yml`, has 0 errors, 0 warnings, and 0 info messages. It also shows the encoding as UTF-8, the format as YAML, and the git status as master with no remote.

Validation Tools

Tools that check or "lint" your file.

- Speccy is a CLI tool with configurable rules <http://speccy.io/>
- Open API Spec Validator
<https://github.com/plc2u/openapi-spec-validator>

Set up in your editor or use a watch command, e.g.:

```
watch -n 1 speccy lint myspec.yml
```

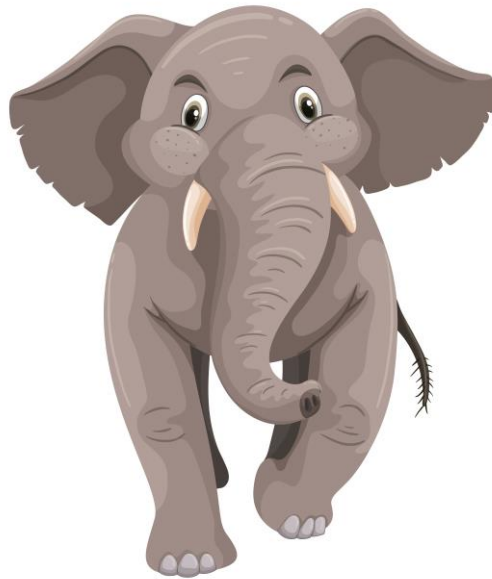
Speccy is opinionated. Mostly in a good way,
but some rules off in do the heavy lifting along the
way!

Preview Tools

OAS is a standard! So any preview should do:

- ReDoc is great <https://github.com/Rebilly/ReDoc>
- Speccy also wraps ReDoc for its `serve` command
- You can run OpenApi-GUI locally
<https://github.com/mermade/openapi-gui>

Creating OpenAPI Specs is like eating an elephant



One bite at a time! Build it up in steps, use the tools

@lornajane

Uses for OpenAPI Spec



@lornajane

Resources

- <https://www.openapis.org>
- <https://apievangelist.com>
- <https://speccy.io>
- <https://github.com/Rebilly/ReDoc>
- <https://openapi.tools>
- <https://github.com/openapitools/openapi-generator>

The magic is the unknown a year's answers
on postcard (or twitter) [openapi.tools](#)

@lornajane

Bonus Extra Slides

@lornajane

Code Generators

Libraries can be generated as easily as docs.

For PHP (and so many other languages) try
<https://github.com/openapitools/openapi-generator>

Pull docker image, generate PHP code from your OpenAPI Spec

Code Generator Example

```
1 $config->setUsername(NEXMO_API_KEY);
2 $config->setPassword(NEXMO_API_SECRET);
3
4 $client = new OpenAPI\Client\Api\DefaultApi(
5     new GuzzleHttp\Client(), $config);
6 $obj = new \OpenAPI\Client\Model\InlineObject();
7
8 try {
9     $result = $client->retrieveSecrets(NEXMO_API_KEY, $obj);
10    print_r($result);
11 } catch (Exception $e) {
12     echo 'Exception when calling DefaultApi->retrieveSecrets: ', $e->getMessa
13 }
```

Code Generator Example

```
1 $config->setUsername(NEXMO_API_KEY);
2 $config->setPassword(NEXMO_API_SECRET);
3
4 $client = new OpenAPI\Client\Api\DefaultApi(
5     new GuzzleHttp\Client(), $config);
6 $obj = new \OpenAPI\Client\Model\InlineObject();
7
8 try {
9     $result = $client->retrieveSecrets(NEXMO_API_KEY, $obj);
10    print_r($result);
11 } catch (Exception $e) {
12     echo 'Exception when calling DefaultApi->retrieveSecrets: ', $e->getMessage();
13 }
```

For a lib, I might wrap the calls so I could add
some ~~bits~~ if needed ~~with~~ etc

@lornajane