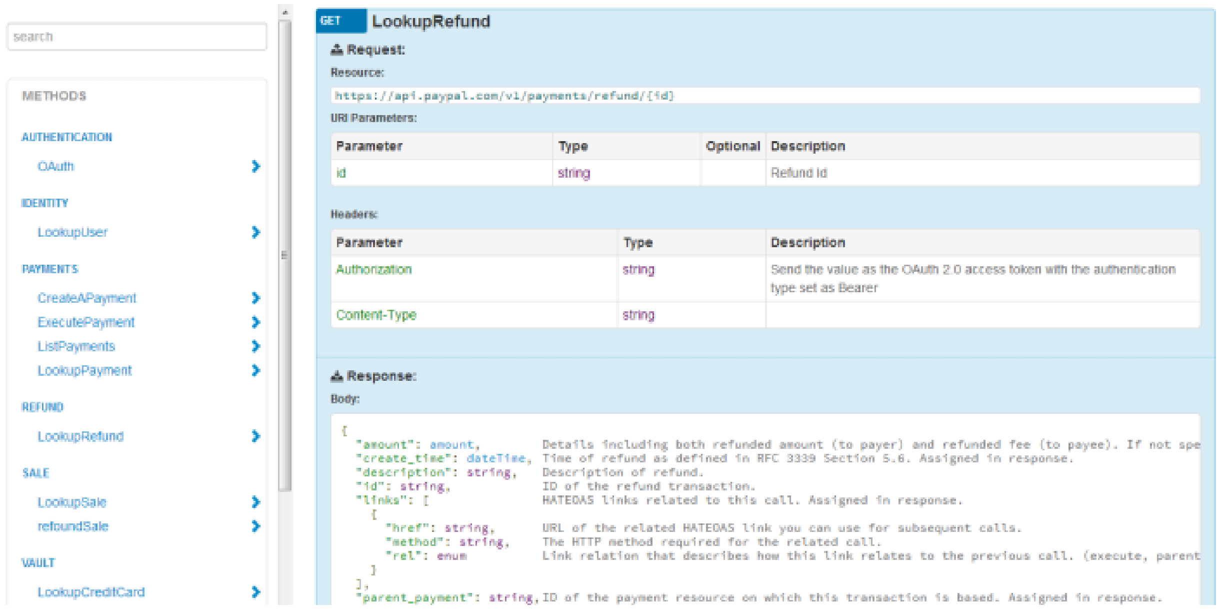
ApiDoc Documentation

Release 1.4

Jeremy Derusse

Nov 12, 2018

Contents



LookupRefund

Request:

Resource:

```
https://api.paypal.com/v1/payments/refund/{id}
```

URI Parameters:

Parameter	Type	Optional	Description
id	string		Refund Id

Headers:

Parameter	Type	Description
Authorization	string	Send the value as the OAuth 2.0 access token with the authentication type set as Bearer
Content-Type	string	

Response:

Body:

```
{
  "amount": amount,        Details including both refunded amount (to payer) and refunded fee (to payee). If not spe
  "create_time": dateTime, Time of refund as defined in RFC 3339 Section 5.6. Assigned in response.
  "description": string,   Description of refund.
  "id": string,            ID of the refund transaction.
  "links": [               HATEOAS links related to this call. Assigned in response.
    {
      "href": string,      URL of the related HATEOAS link you can use for subsequent calls.
      "method": string,    The HTTP method required for the related call.
      "rel": enum          Link relation that describes how this link relates to the previous call. (execute, parent
    }
  ],
  "parent_payment": string,ID of the payment resource on which this transaction is based. Assigned in response.
```

CHAPTER 1

Summary

ApiDoc is a documentation generator designe for API built with Python and given by SFR Business Team.

ApiDoc consists of a command line interface. It is maintained in a single repository. By using this application you automatically require all of the necessary modules dependencies which are:

- Demo: http://solutionscloud.github.io/apidoc/demo
- Home Page: http://solutionscloud.github.io/apidoc
- Documentation: http://apidoc.rtfd.org
- Bug Tracker: https://github.com/SolutionsCloud/apidoc/issues
- GitHub: https://github.com/SolutionsCloud/apidoc
- PyPI: https://preview-pypi.python.org/project/ApiDoc
- License: GPLv3+

CHAPTER 2

Requirements

For core application

- PyYAML
- Jinja2
- JsonSchema

For developers who want to contribute code to ApiDoc

- behave
- coverage
- mock
- nose
- yuicompressor
- Sphinx

CHAPTER 3

Contents

3.1 Quick Start

3.1.1 Installation

The fastest way to get started is by using the command line tool

```
$ sudo apt-get install python3-pip
$ sudo pip3 install apidoc
```

If the package python3-pip does not exists.

```
$ sudo apt-get install python3-setuptools
$ sudo easy_install3 pip
$ sudo pip3-2 install apidoc
```

The config parser script depends on PyYAML which links with LibYAML, which brings a performance boost to the PyYAML parser. However, installing LibYAML is optional but recommended. On Mac OS X, you can use homebrew to install LibYAML:

```
$ brew install libyaml
```

On Linux, use your favorite package manager to install LibYAML. Here's how you do it on Debian/Ubuntu:

```
$ sudo apt-get install libyaml-dev python3-dev
```

On Windows, please install PyYAML using the binaries they provide

3.1.2 Run a sample demo

```
$ mkdir apidoc
$ cd apidoc
$ wget https://raw.github.com/SolutionsCloud/apidoc/master/example/demo/source.yml
$ apidoc -i source.yml -o output/index.html
$ firefox output/index.html
```

3.2 Usage of ApiDoc

3.2.1 Main commands

apidoc builds the full documentation

```
$ apidoc -h
```

3.2.2 Generics Arguments

To generate documentation from a given source file:

```
$ apidoc -i ./example/source_simple/simple.yml
```

see page *Source's File Format*

To generate documentation from split sources in multiple files:

```
$ apidoc -i ./example/source_multiple/one.yml ./example/source_multiple/two.yml
```

see page *Source's File Format*

To generate documentation from the files contained in a given directory:

```
$ apidoc -i ./example/source_multiple/
```

see page *Source's File Format*

To generate documentation with options defined in a given config file:

```
$ apidoc -c ./example/config/config.yaml
```

see page *Config's File Format*

Combining those options:

```
$ apidoc -c ./config.yaml -i ./folder1/ ./folder2/ /folder3/file.yaml /folder3/file.
↪json
```

Analyse the sources files without buiilding the documentation:

```
$ apidoc -i ./example/source_simple/simple.yml -y
```

Render automaticly the documentation each time a file is changed:

```
$ apidoc -i ./example/source_simple/simple.yml -w
```

Display less logging informations

```
$ apidoc -i ./example/source_simple/simple.yml -q
$ apidoc -i ./example/source_simple/simple.yml -qq
```

Display traceback (for advanced users)

```
$ apidoc -i ./example/source_simple/simple.yml -t
```

3.3 Config's File Format

ApiDoc can be used with arguments in the command line or with a config file containing the list of options (or both arguments and config file)

The format of the config file can be either *YAML* or *JSON*, the extension of the file must be respectively *.yaml* (or *.yml*) or *.json*

3.3.1 Usage

To use ApiDoc with a config file call the following arguments :

```
$ apidoc -c ./path-to-config.yaml
```

3.3.2 Sample

This is a minimalistic sample of a config file

```
input:
  locations:
    - ./sources/one.yml
output:
  location: ./output/sample.html
```

Here is a basic sample of a config file

```
input:
  locations:
    - ./sources
    - ./sources2/one.yml
  arguments:
    url: api.sfr.com
filter:
  versions:
    excludes:
      - v2
output:
  location: ./output/sample.html
  componants: local
  template: default
```

3.3.3 input

The section input defines where the source files are located. It contains three sub sections *locations validate* and *arguments*. The first subsection contains a list of directories or files and the third a list of arguments (or variables) which will be used by the source files (see *Variables*). The validate flag define if the sources files should be validate by the json schema validator. As for config files, the extensions of source files must be *.yaml* (or *.yml*) or *.json*. When a directory is specified in the *locations* subsection, all the source files (with a valid extension) contained in the directory (or in a sub directory) will be merged into a single virtual source file which will be used to generate the documentation (see *Source's File Format*). A config file must reference at least one input source file.

This is a full sample of the section input

```
input:
  locations:
    - ./project/api-sources
    - ../common-api/sources
    - ./project/api-v2-source/demo.yaml
    - ./project/api-v2-source/common.yaml
  validate: False
  arguments:
    url: api.sfr.com
    defaultVersion: v1
```

3.3.4 filter

The section filter provides a way to exclude or include versions and/or category in the rendered documentation. This section contains two sub sections : *versions* and *categories* which both contain two subsections *includes* and *excludes*. To include a specifique list of versions (or categories) and ignoring the others, specify these versions (or categories) in the *includes* subsection. To ignore a specific list of versions (or categories) and including the others, specify these versions (or categories) in the *exclude'subsection. If the 'filter* section is missing (or empty), all versions and sections will be displayed. If the *versions* (or *categories*) subsection is missing (or empty), all versions (or categories) will be displayed. If the *includes* subsection is missing (or empty), all but excluded versions (or categories) will be displayed. If the *excludes* subsection is missing (or empty), no versions (or categories) will be removed.

The excluded versions (and categories) will be removed at the end of the rendering process. If a displayed version (or category) extends an ignored version (or category), this version will be displayed normally.

Here is a full sample of a section filter

```
filter:
  versions:
    includes:
      - v1.0
      - v2.0
  categories:
    excludes:
      - Experiment
      - Draft

filter:
  versions:
    excludes:
      - v3.0
  categories:
    include:
```

(continues on next page)

(continued from previous page)

```
      - Authentication
      - Common
```

3.3.5 output

The section describes the format and the location of the rendered documentation. It contains three subsections: *location*, *template* and *componants*. The *location* subsection defines the relative (or absolute) path to the file where ApiDoc will generate the documentation. When the value is *stdout* the rendered result will be display on the standard output of the console. (Beware of using this mode with the command *analyse-watch*) The *template* subsection defines the relative (or absolute) path to the template used to render the documentation. ApiDoc uses the template engine Jinja, for a full documentation see the official site. When the value is *default* ApiDoc will use the default template. The *componants* subsection defines where the assets (css, javascripts, images, fonts) are stored. The possible values are:

- *local*: The files are stored in the same folder as the output
- *embedded*: The files are embedded in the generated documentation
- *remote*: the generated documentation will reference remote assets using CDN or public repositories
- *without*: The files are not generated in documentation

The *layout* subsection defines the layout used by default template. The possible values are:

- *default*: Standard layout with header
- *content-only*: Layout without headers

This is a full sample of the section ouput

```
output:
  location: ./project/documentation.html
  componants: ./project/template/custom.html
  template: default
  layout: default
```

3.4 Source's File Format

ApiDoc uses source file(s) to generate the documentation. The format of the source file can be *YAML* or *JSON*, the extension of the files must be repectively *.yaml* (or *.yml*) or *.json*

This is a basic sample of a config file

```
configuration:
  title: Hello API
  description: An API dedicated to the hello service
  uri: ${base_url}
categories:
  Display:
    description: Display messages
  Config:
    description: Configure the application
versions:
  v1.0:
    methods:
      Hello:
```

(continues on next page)

(continued from previous page)

```
        category: Display
        uri: /
        description: Say hello
        response_body:
          type: string
          sample: Hello
      HelloName:
        category: Display
        uri: /{name}
        request_parameters:
          name:
            type: string
            description: Name of the user
        request_headers:
          Accept:
            type: mimeType
            description: List of accepted MimeTypes
            sample: text/plain
        response_body:
          type: string
          sample: Hello my_name
      ConfigHello:
        category: Config
        uri: /
        method: PUT
        description: Configure the hello method
        request_body:
          type: object
          properties:
            language:
              type: string
              sample: "fr"
        response_body:
          type: boolean
    types:
      mimeType:
        item:
          type: string
          sample: application/json
        format:
          pretty: type/sous-type
  v2.O:
    status: beta
    display: false
```

All elements are optional, but a least one displayable method is required.

3.4.1 configuration

The *configuration* contains major information of your documentation.

- uri: Common URI of your API (ie: https://api.sfr.com/service/)
- title: The name of your API. It will be displayed in the title tag, and in the header of the documentation
- description: A description of your API. It will be displayed under the title in the header of the documentation

sample:

```
configuration:
  title: Hello API
  description: An API dedicated to the hello service
  uri: https://api.sfr.com/services/hello
```

3.4.2 versions

This element contains a dictionary of versions. Each version is associated with a key which is the name of the version. At least one version is required.

- display: Boolean Defining if the version is displayed or not.
- label: The label of the version who will be display in the documentation.
- uri: Completes the URI of the element *configuration*.
- major: Major part of the version number.
- minor: Minor part of the version number.
- status (*current*, *beta*, *deprecated*, *draft*): Status of the version.
- methods: List of methods contained in the version (see *methods*).
- types: List of types contained in the version (see *types*).
- references: List of references contained in the version (see *references*).

sample:

```
versions:
  v1.0:
    display: true
    label: Version 1
    uri: /v1
    major: 1
    minor: 0
    status: current
    methods:
      ...
  v2.0:
    display: false
    uri: /v2
    major: 2
    minor: 0
    status: beta
    methods:
      ...
```

3.4.3 categories

This element contains a dictionary of categories; a category is a kind of folder in the rendered documentation. Each category is associated to a key which is the name of the category. This elements is not required. A method (or a type) can have an attribute category which does not exist in this dictionnary. But this element allows you to configure the category by giving a description or a priority order.

- display: Boolean defining if the category is displayed or not

- label: The label of the category who will be display in the documentation
- description: A description of the category. It will be dislayed under the name of the category
- order: A number used to sort the categories and define in which order they will be displayed. Default value is 99. When two categories have the same order, they will be sorted alphabetically by name.

sample:

```
categories:
  Common:
    display: false
    description: a helper section use for extension
  Version:
    description: List the version of the API
    order: 1
  Authentication:
    label: Authentication + Logout
    description: How to login and logout the client
```

3.4.4 methods

This element contains a dictionary of methods. Each method is associated with a key which is the name of the method. At least one method is required.

- label: The label of the method who will be display in the documentation.
- description: A description of the method. It will be dislayed under the name of the method.
- uri: Endpoint of the method based on the URI found in *configuration* and *version*. Parameters are declared between curly bracket.
- method (*get*, *post*, *put*, *delete*, *head*, *http*): Type of method used in this endpoint (default *get*).
- code: Normal code returned by the method (default *200*). This information will be displayed in the sample generated in the documentation.
- request_parameters: List of parameters sent in the URI of the method (see *request_parameters*).
- request_headers: List of parameters sent in the headers of the method (see *request_headers*).
- request_body: Request object sent in the body of the method (see *request_body*).
- response_codes: List of codes received in the headers of the method (see *response_codes*).
- response_body: Response object received in the body of the method (see *response_body*).
- category: The name of the category to which the method belongs.

sample:

```
methods:
  Hello:
    label: Echo
    uri: /hello-{name}.json
    method: get
    code: 200
    description: Say hello
    request_parameters:
      ...
    request_headers:
```

(continues on next page)

(continued from previous page)

```
      ...
    request_body:
      ...
    response_codes:
      ...
    response_body:
      ...
```

request_parameters

This element contains a dictionary of query parameters contained in the URI. Each parameter is associated with a key which is the name of the parameter. If a parameter is defined in this elements but is not on the URI, it will not be displayed. This can be usefull when you use extensions. The parameters will be displayed in the same order as they appear in the URI

- type: Type of the parameter (see *types*).
- description: A description of the parameter.
- optional: A boolean indicating if the parameter is compulsory or optional. Default False.
- sample: A sample value which will be displayed in the sample fieldset.
- generic: If true, the parameter will be displayed in an other color. Default False.

sample:

```
request_parameters:
  name:
    type: string
    description: Name of the user
    optional: false
    sample: John Doe
    generic: false
  language:
    type: string
    description: Language of the response
    optional: true
    sample: fr
    generic: true
```

request_headers

This element contains a dictionary of header parameters expected by the method. Each parameter is associated with a key which is the name of the parameter.

- type: Type of the parameter (see *types*).
- description: A description of the parameter.
- optional: A boolean indicated if the parameter is compulsory or optional.
- sample: A sample value which will be displayed in the sample fieldset.
- generic: If true, the parameter will be displayed in an other color. Default False.

sample:

```
request_headers:
  Accept:
    type: mimeType
    description: List of accepted MimeTypes
    sample: text/plain
    optional: true
    generic: true
  X-Auth-Token:
    type: string
    description: Authentication token
```

request_body

This element contains an object which represents the raw content sent in the body of the request (see *Objects*).

sample:

```
request_body
    type: object
    description: Root envelope
    properties:
      login:
        type: string
      password:
        type: string
```

response_codes

This element contains a list of reponse codes returned by the method.

- code: The numeric code returned in the response
- message: A message associated to the response. When omitted, the default message associated with the code will be used
- description: A description of the response
- generic: If true, the parameter will be displayed in an other color. Default False

sample:

```
response_codes:
- code: 400
  message: Bad name format
  description: The resource name is not correct
- code: 404
  message: Resource not found
  generic: true
```

response_body

This element contains an object which represents the response received in the body of the response (see *Objects*).

sample:

```
request_body
  type: array
  description: List of users
  items:
    type: object
    properties:
      name:
        type: string
        description: Name of the user
      role:
        type: CustomRole
        description: Role of the user
```

3.4.5 types

This element contains a dictionary of types used by other elements. Each reference is associated with a key which is the name of the type. When a type is declared but never used, a warning will be fired in the logs and the type will not be displayed.

- category: The name of the category to which the type belongs.
- description: A description of the type.
- item: The content of the type (see *Objects*).
- **format: Some representations of the type.**
  - pretty: A well formated representation of the type.
  - advanced: A technically accurate representation of the type.

sample:

```
types:
  mimeType:
    category: Common
    description: A mime type
    item:
      type: string
      sample: application/json
    format:
      pretty: type/sous-type
      advanced: [a-z]\/[a-z]
  languages:
    category: Lists
    description: List of supported language
    item:
      type: enum
      values:
      - en
      - fr
      descriptions:
        en:
          description: English
        fr:
          description: Français
```

3.4.6 references

This element contains a dictionary of references used by objects. Each reference is associated with a key which is the name of the reference. The reference is not displayed directly, it is a complex object which could be used in other elements.

sample:

```
methods:
  listComments:
    ...
    response_body:
      type: array
      items:
        type: reference
        reference: comment
reference:
  comment:
    type: object
    properties:
      owner:
        type: reference
        reference: user
      message:
        type: string
      date:
        type: string
  user:
    type: object
    name:
      type: string
    language:
      type: string
```

3.4.7 Objects

In the bodies of types, requests and responses you can define a complex object using basic elements. These elements (defined below) contain always a keyword "type" which defines the type of the element. The known types, are *object*, *array*, *dynamic*, *boolean*, *none*, *string*, *number*, *integer*, *reference*, *const*, *enum*. If the type is not in this list, ApiDoc will look in the elements declared in the *types* section (see *types*). Each elements contains an attribute *optional* indicating if the element is compulsory or optional. They also contains an attribute *constraints* containing a dictionnary constraints. Some constraints are predefined depending of the type of the element, but it"s also possible to define custom constraints (a reference to yout business model for example). No check will be applied on sample according to these constraints, they only will be display in a popover in the rendered documentation. These constraints are derived from Json Schema

String

The object String defines a string.

- description: A description of the string
- sample: A sample value which will be displayed in the sample fieldset.
- optional: A boolean indicating if the parameter is compulsory or optional. Default False.
- maxLength: A positive integer is expected.

- minLength: A positive integer is expected.
- pattern: A string is expected. It who should be a regular expression, according to the ECMA 262 regular expression dialect.
- format: A string is expteced. It must be one of the values found in this list (date-time, email, hostname, ipv4, ipv6, uri).
- enum: An array of string is expected.
- default: A string is expected.
- constraints: A dictionary of constraints * {constraint_name}: A string is expected

sample:

```
name:
  type: string
  description: Name of the user
  sample: John Doe
  minLength: 1
  maxLength: 32
```

Number

The object Number defines a numeric value with optionals decimals.

- description: A description of the number
- sample: A sample value which will be displayed in the sample fieldset.
- optional: A boolean indicating if the parameter is compulsory or optional. Default False.
- multipleOf: A number is expected.
- maximum: A number is expected.
- exclusiveMaximum: A boolean is expected.
- minimum: A number is expected.
- exclusiveMinimum: A boolean is expected.
- enum: An array of number is expected.
- default: A number is expected.
- constraints: A dictionary of constraints * {constraint_name}: A string is expected

sample:

```
price:
  type: number
  description: Price in dollars
  sample: 20.3
  maximum: 0
  multipleOf: 0.01
```

Integer

The object Integer defines a numeric value without decimal.

- description: A description of the number
- sample: A sample value which will be displayed in the sample fieldset.
- optional: A boolean indicating if the parameter is compulsory or optional. Default False.
- multipleOf: A integer is expected.
- maximum: A integer is expected.
- exclusiveMaximum: A boolean is expected.
- minimum: A integer is expected.
- exclusiveMinimum: A boolean is expected.
- enum: An array of integer is expected.
- default: A integer is expected.
- constraints: A dictionary of constraints * {constraint_name}: A string is expected

sample:

```
age:
  type: number
  description: Age of the user
  sample: 20
  maximum: 0
```

Boolean

The object Boolean defines a boolean.

- description: A description of the boolean
- sample: A sample value which will be displayed on the sample fieldset.
- optional: A boolean indicating if the parameter is compulsory or optional. Default False.
- default: A boolean is expected.
- constraints: A dictionary of constraints * {constraint_name}: A string is expected

sample:

```
is_default:
  type: boolean
  description: Define if the group is the default group
  sample: false
  default: false
```

None

The object None defines an empty object. Sometime used in a request when a key is compulsory but no value is expected.

- description: A description of the object
- optional: A boolean indicating if the parameter is compulsory or optional. Default False.
- constraints: A dictionary of constraints * {constraint_name}: A string is expected

sample:

```
reboot:
  type: none
  description: Set this key if you want reboot your server
  constraints:
    compulsory: yes
```

Const

The object Const defines an constant property. Sometime used in a request like the property "method" in Json-RPC.

- description: A description of the object
- cont_type: A scalar type of the constant (allowed values are *string*, *number*, *integer*, *boolean*). If undefined *string* will be used
- value: The value associated to the property
- optional: A boolean indicating if the parameter is compulsory or optional. Default False.
- constraints: A dictionary of constraints * {constraint_name}: A string is expected

sample:

```
method:
  type: const
  description: Json-RPC method name
  const_type: string
  value: "find"
  constraints:
    required: authenticated user
```

Enum

The object Enum defines a list a availables values. When this object is the primary object of an type (see *types*) the values with there descriptions will be displayedin the Type section.

- description: A description of the object
- values: An array of values
- descriptions: A dictionnary of description for each value
- optional: A boolean indicating if the parameter is compulsory or optional. Default False.
- constraints: A dictionary of constraints * {constraint_name}: A string is expected

sample:

```
httpMethods:
  type: enum
  description: List of Http methods used in Rest
  values:
  - GET
  - POST
  - PUT
  - DELETE
  descriptions:
```

(continues on next page)

(continued from previous page)

```
    GET: Like select
    POST: Like insert
    PUT: Like update
    DELETE: Like delete
  sample: GET
  constraints:
    required: authenticated user
```

Object

The object Object defines a complex object containing a dictionnary of properties, patternProperties or additionalProperties. Each property is associated with a key which is the name of the property.

- description: A description of the object
- properties: List of properties of the object
- patternProperties: List of properties of the object where the key is a regular expression
- additionalProperties: A boolean False when additional properties are not allowed, Otherwise it's an object
- optional: A boolean indicating if the parameter is compulsory or optional. Default False.
- constraints: A dictionary of constraints * {constraint_name}: A string is expected

sample:

```
element:
  type: object
  description: User to update
  properties:
    name:
      type: string
      description: New name of the user
    metadata:
      type: object
      additionalProperties:
        type: string
  constraints:
    required: authenticated user
```

Array

The object Array defines an array of objects.

- description: A description of the array
- items: A representation of the items contained in the array
- sample_count: Number of items to display in the sample fieldset
- optional: A boolean indicating if the parameter is compulsory or optional. Default False.
- maxItems: A positive integer is expected.
- minItems: A positive integer is expected.
- uniqueItems: A boolean is expected.
- constraints: A dictionary of constraints * {constraint_name}: A string is expected

sample:

```
elements:
  type: array
  description: List of users
  items:
    type: object
    properties:
      name:
        type: string
        description: New name of the user
  maxItems: 10
```

Reference

The object Reference defines a reference to a referenced object. Reference is the only one elements who does not have constraints. This constraints are defined in the refererenced item.

- reference: Name of the reference
- optional: A boolean indicating if the parameter is compulsory or optional. Default False.

sample:

```
user:
  type: reference
  reference: GenericUser
```

Dynamic (deprecated)

The object Dynamic defines a special object where the key, which must be a string, is dynamic. You should use object with additionalProperties instead of this dynamic element.

- description: A description of the array
- items: A representation of the items contained in the object
- sample: A sample value which will be displayed on the sample fieldset.
- optional: A boolean indicating if the parameter is compulsory or optional. Default False.
- maxItems: A positive integer is expected.
- minItems: A positive integer is expected.
- constraints: A dictionary of constraints * {constraint_name}: A string is expected

sample:

```
metadatas:
  type: dynamic
  description: A list of key/value to store what you want
  item: string
```

3.4.8 Customizations

Variables

Variables can be provided by command line arguments or in the config file (see *input*). Each value of the source file can be (or contain) a variable.

Sample using a variable string context:

```
configuration:
  title: ${applicationName}
  description: Official documentation of ${applicationName}
```

Sample using a variable to set a boolean:

```
categories:
  MyExperiments:
    display: ${displayExperimentals}
```

Sample using a variable used in extends context:

```
versions:
  v1:
    ...
  v2:
    extends: ${officialVersion}
```

3.4.9 Extends

ApiDoc provides a way to simplfy source files writing by using an extends system. You can extend your *versions*, *categories*, *methods*, *types* and *references*. To extend an element you only have to specify the name of referenced elements with *extends: referenced_name*. The referenced name is always of the same type as your elements (a *version* extends a *version*, a *method* extends a *method*, etc. . .). If the referenced element is a sibling of the current element, you can just specify its name, if the reference does not belong to the same parent, you must then specify the name of the parent followed by a */*. For exemple, if your methods A and B belong to different versions you must use *extends: version_of_b/method_b* for method A to extend method B. Extensions are recursive, but you can break recursion on any element by using *inherit: false* and you can remove an element with *removed: true*.

Sample using an extension on the version:

```
versions:
  v1:
    ...
  v2:
    extends: v1
```

Sample using an extension on a method with relative path and absolute path:

```
versions
  v1:
    methods:
      Request:
        ...
      AuthenticatedRequest:
        extends: Request
  v2:
```

(continues on next page)

(continued from previous page)

```
    methods:
      Login:
        extends: v1/Request
```

Sample using an extension where the method ListClientWithDetails extends ListClients but not for the response_body which is redefined:

```
methods:
  ListClients:
    ...
  ListClientWithDetails:
    extends: ListClients
    response_body:
      inherit: false
    ...
```

Sample using an extension where the section SessionAuthentication extends FormAuthentication but the content of the body of the method Login is removed:

```
methods:
  Login:
    request_body:
      type: object
      properties:
        login:
          type: string
        password:
          type: string
  SSO:
    extends: Login
    request_parameters:
      token_id:
        type: string
    request_body:
      removed: true
```

You can extend multiple elements by providing an list of extensions.

```
Methods:
  Authenticated:
    request_header:
      X-Auth-Token:
        type: string
  Paginated:
    request_parameter:
      index:
        type: integer
      limit:;
        type: integer
  Customers:
    extends:
    - Authenticated
    - Paginated
```

3.5 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

3.5.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/SolutionsCloud/apidoc/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with “enhancement” is open to whoever wants to implement it.

Write Documentation

ApiDoc could always use more documentation, whether as part of the official ApiDoc docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/SolutionsCloud/apidoc/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

3.5.2 Setting Up the Code for Local Development

Here’s how to set up *ApiDoc* for local development.

1. Fork the *ApiDoc* repo on GitHub.
2. Clone your fork locally

```
$ git clone git@github.com:your_name_here/apidoc.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development

```
$ mkvirtualenv apidoc
$ cd apidoc/
$ pip install -e .[contribute]
```

4. Create a branch for local development

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass the tests and flake8

```
$ flake8 --show-source --ignore=E501 --statistics .
$ python setup.py test
```

6. Commit your changes and push your branch to GitHub

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Check that the test coverage hasn't dropped

```
$ behave --format progress2 tests/features/
$ coverage3 run --branch --source apidoc setup.py test
$ coverage3 report -m
```

8. Submit a pull request through the GitHub website.

3.5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.2, 3.3. Check https://travis-ci.org/SolutionsCloud/apidoc/pull_requests and make sure that the tests pass for all supported Python versions.

3.5.4 Tips

To run a particular test

```
$ python -m unittest tests.test_find.TestFind.test_find_template
```

To run a subset of tests

```
$ python -m unittest tests.test_find
```

CHAPTER 4

Licenses

ApiDoc uses the following projects:

Twitter Bootstrap

Jquery

MouseTrap

Icon Minia

Entypo

IcoMoon