Flex Documentation

Release 6.4.0

Piper Merriam

Mar 31, 2017

Contents

Flex is a Swagger 2.0 validator. It is currently at a very early stage in development and thus is likely to have plenty of bugs.

CHAPTER 1

General Use

install `flex`

```
$ pip install flex
```

Then in your code.

```
import flex

schema = flex.load('path/to/schema.yaml')
```

CHAPTER 2

Supported Schema Formats

The `flex.load` function supports the following.

- A url to either a `json` or `yaml` schema.
- A path to either a `json` or `yaml` schema.
- A file object whose contents are `json` or `yaml`
- A string whose contents are `json` or `yaml`
- A native python object that is a `Mapping` (like a dictionary).

CHAPTER 3

JSON Schema Validation

A subset of the `flex` tooling implements JSON schema validation.

```
>>>  from flex.core import validate
>>>  schema = {
...:   'properties': {
...:     'name': {
...:       'type': 'string',
...:       'minLength': 3,
...:     },
...:     'age': {
...:       'type': 'integer',
...:       'minimum': 0,
...:       'exclusiveMinimum': True,
...:     },
...:   }
...: }
>>>  data = {
...:   'age': 10,
...:   'name': "John",
...: }
>>>  validate(schema, data)
>>>  bad_data = {
...:   'age': -5,
...:   'name': "Bo",
...: }
>>>  validate(schema, bad_data)
ValueError: Invalid:
'age':
    - 'minimum':
        - u'-5 must be greater than than 0.0'
'name':
    - 'minLength':
        - u'Ensure this value has at least 3 characters (it has 2).'
```

You can also use this to simply validate that your JSON schema conforms to specification.

```
>>>  from flex.core import validate
>>>  schema = {
...:    'properties': {
...:      'name': {
...:        'type': 'string',
...:        'minLength': 3,
...:      },
...:      'friends': {
...:        'type': 'array',
...:        'minimum': 0,  # `minimum` is invalid for type 'array'
...:      },
...:    }
...: }
>>>  validate(schema)
ValueError: JSON Schema did not validate:

u'properties':
    - 'friends':
        - 'minimum':
            - u'`minimum` can only be used for json number types'
```

CHAPTER 4

API Call Validation

API call validation takes a supported request and response object that represents a request/response cycle for an API call and validates it against a swagger schema.

```
>>> import requests
>>> from flex.core import load, validate_api_call
>>> schema = load("path/to/schema.yaml")
>>> response = requests.get('http://www.example.com/api/')
>>> validate_api_call(schema, raw_request=response.request, raw_response=response)
ValueError: Invalid
'response':
    - 'Request status code was not found in the known response codes.  Got `301`:␣
↪Expected one of: `[200]`'
```

Request validation looks at the following things.

1. Request path.
2. Parameters in the path.
3. Query parameters.
4. Request method.
5. Headers.
6. Content Type

Response validation looks at the following things.

1. Content-Type
2. Headers
3. Status Code
4. Response body.

Request validation supports the following request objects.

- `requests.Request` and `requests.PreparedRequest` from Kenneth Reitz' `requests` library.

- `urllib2.Request` from the `urllib2` module of the standard library.

Response valdation supports the following respone objects.

- `requests.Response` from Kenneth Reitz' `requests` library.
- The return value of `urllib.urlopen` and `urllib2.urlopen` from the standard library urllib modules.

CHAPTER 5

Formats

Flex implements format validation for the following formats

- `uuid`: Version 1, 3, 4, and 5
- `datetime`: iso8601 formatted datetimes via https://pypi.python.org/pypi/iso8601.
- `int32`: Integers up to 32 bits.
- `int64`: Integers up to 64 bits.
- `email`: via https://pypi.python.org/pypi/validate_email
- `uri`: via https://pypi.python.org/pypi/rfc3987

Flex supports registering your own custom formats for validation.

```
>>> from flex.formats import register
>>> @register('title-case', 'string')
... def title_case_format_validator(value):
...     if not value == value.title():
...         raise ValidationError("Must be title cased")
```

In the example above, we have registered a new format `title-case` which is applicable to values of type string. A validator function needs to take a single value and raise a `ValidationError` if the value is invalid.

The `register` decorator takes the name of the format as it's first argument, and then the remaining arguments should be the types that the format validator can apply to.

Note: Take note that format validation is skipped if the value is not of one of the specified types the format validator is declared for.

CHAPTER 6

Command line usage

As well as a python API, `flex` also provides a commandline validation tool via the `swagger-flex` cli.

```
$ ./swagger-flex -s /path/to/swagger.yaml
$ ./swagger-flex -s http://spec.example.com/swagger.yaml
```

In the event of a validation error, the commandline program will return 1 and print to stderr a list of the validation errors detected.

If the file passes validation it will return to stdout "Validation passed" and return 0 - in line with most *nix commandline tools*.

Contents:

CHAPTER 7

Indices and tables

- genindex
- modindex
- search