bravado$_{core}$

Release 5.10.2

Mar 07, 2019

Contents

bravado_core is a Python library that implements the Swagger 2.0 Specification.

Client and servers alike can use bravado_core to implement these features:

- Swagger Schema ingestion and validation
- Validation and marshalling of requests and responses
- Validation and marshalling of user-defined Swagger formats
- Modelling Swagger *#/definitions* as Python classes or dicts

For example:

- bravado uses bravado-core to implement a fully functional Swagger client.
- pyramid_swagger uses bravado-core to seamlessly add Swagger support to Pyramid webapps.

Contents:

CHAPTER 1

Configuration

All configuration is stored in a `dict`.

```python
from bravado_core.spec import Spec

spec_dict = json.loads(open('swagger.json', 'r').read())

config = {
    'validate_requests': False,
    'use_models': False,
}

swagger_spec = Spec.from_dict(spec_dict, config=config)
```

Config key	Type	Default	Description
validate_swagger_spec	boolean	True	Validate the Swagger spec against the Swagger 2.0 Specification.
validate_requests	boolean	True	On the client side, validates outgoing requests. On the server side, validates incoming requests.
validate_responses	boolean	True	On the client side, validates incoming responses. On the server side, validates outgoing responses.
use_models	boolean	True	Use python classes to represent models instead of dicts. See *Python Models*.
formats	list of SwaggerFormat	[]	List of user-defined formats to support. See *User-Defined Formats*.
include_missing_properties	boolean	True	Create properties with the value `None` if they were not submitted during object unmarshalling
default_type_to_object	boolean	False	When set to `True`, missing types will default to `object` and be validated as such. When set to `False`, missing types will not be validated at all.
internally_dereference_refs	boolean	False	Completely dereference $refs to maximize

Python Models

Models in a Swagger spec are usually defined under the path `#/definitions`.

A model can refer to a primitive type or a container type such as a `list` or a `dict`. In `dict` form, there is an opportunity to make the interface to access the properties of a model a little more straight forward.

Consider the following:

```json
{
    "definitions": {
        "Pet": {
            "type": "object",
            "required": ["name"],
            "properties": {
                "name": {"type": "string"},
                "age": {"type": "integer"},
                "breed": {"type": "string"}
            }
        }
    }
}
```

In python, this model easily maps to a `dict`:

```python
pet = {
    "name": "Sumi",
    "age": 12,
    "breed": None,
}

print pet['name']

if pet['age'] < 1:
    print 'What a cute puppy!'
```

```python
if pet['breed'] is None:
    pet['breed'] = 'mutt'
```

However, if the model is implemented as a Python type, dotted access to properties becomes a reality:

```python
from bravado_core.spec import Spec

spec = Spec.from_dict(...)
Pet = spec.definitions['Pet']
pet = Pet(name='Sumi', age=12)

print pet.name

if pet.age < 1:
    print 'What a cute puppy!'

if pet.breed is None:
    pet.breed = 'mutt'
```

2.1 Configuring Models as Python Types

bravado-core supports models as both dicts and python types.

The feature to use python types for models is enabled by default. You can always disable it if necessary.

```python
from bravado_core.spec import Spec
swagger_dict = {..}
spec = Spec.from_dict(swagger_dict, config={'use_models': False})
```

2.2 Allowing null values for properties

Typically, bravado-core will complain during validation if it encounters fields with `null` values. This can be problematic, especially when you're adding Swagger support to pre-existing APIs. In that case, declare your model properties as `x-nullable`:

```json
{
    "Pet": {
        "type": "object",
        "properties": {
            "breed": {
                "type": "string",
                "x-nullable": true
            }
        }
    }
}
```

`x-nullable` is an extension to the Swagger 2.0 spec. A `nullable` attribute is being considered for the next major version of Swagger.

2.3 Sensitive Data

Typically, if bravado-core encounters an error validaing a request or a response, the value will be included in the exception message. If you have sensitive data, this can be problematic. To prevent a sensitive value from appearing in the exception details, declare the field as `x-sensitive`:

```json
{
    "Pet": {
        "type": "object",
        "properties": {
            "breed": {
                "type": "string",
                "x-sensitive": true
            }
        }
    }
}
```

`x-sensitive` is an extension to the Swagger 2.0 spec. The `x-sensitive` extension can be applied to arrays and primitives as well as objects.

2.4 Model Discovery

Keep in mind that bravado-core has to do some extra legwork to figure out which parts of your spec represent Swagger models and which parts don't to make this feature work automagically. With a single-file Swagger spec, this is pretty straight forward - almost everything under `#/definitions` is a model. However, with more complicated specs that span multiple files and use external refs, it becomes a bit more involved. For this reason, the discovery process for models is best effort with a fallback to explicit annotations as follows:

1. Search for refs that refer to `#/definitions` in local scope

2. Search for refs that refer to external definitions with pattern `<filename>#/definitions/<model name>`.

swagger.json

```json
{
    "paths": {
        "/pet": {
            "get": {
                "responses": {
                    "200": {
                        "description": "A pet",
                        "schema": {
                            "$ref": "another_file.json#/definitions/Pet"
                        }
                    }
                }
            }
        }
    }
}
```

another_file.json

```
{
    "definitions": {
        "Pet": {
            ...
        }
    }
}
```

3. Search for the `"x-model":  "<model name>"` or `"title":  "<model name>"` annotation to identify models that can't be found via method 1. or 2. In case `x-model` and `title` are both defined, `x-model` has precedence. *swagger.json*

```
{
    "paths": {
        "/pet": {
            "get": {
                "responses": {
                    "200": {
                        "description": "A pet",
                        "schema": {
                            "$ref": "https://my.company.com/definitions/models.
→json#/models/Pet"
                        }
                    }
                }
            }
        }
    }
}
```

models.json (served up via `https://my.company.com/definitions/models.json`)

```
{
    "models": {
        "Pet": {
            "x-model": "Pet"
            ...
        }
    }
}
```

Note: Models will be generated only for object types (`"type":  "object"`).

User-Defined Formats

Primitive types in Swagger support an optional modifier property `format` as explained in detail in the Swagger Specification. With this feature, you can define your own domain specific formats and have validation and marshalling to/from python/json handled transparently.

3.1 Creating a user-defined format

This is best explained with a simple example. Let's create a user-defined format for CIDR notation.

In a Swagger spec, the schema-object for a CIDR would resemble:

```
{
    "type": "string",
    "format": "cidr",
    "description": "IPv4 CIDR"
}
```

In python, we'd like CIDRs to automatically be converted to a `CIDR` object that makes them easy to work with.

```python
class CIDR(object):
    def __init__(self, cidr):
        """
        :param cidr: CIDR in string form.
        """
        self.cidr = cidr

    def overlaps(self, other_cidr):
        """Return true if other_cidr overlaps with this cidr"""
        ...

    def subnet_mask(self):
        """Return the subnet mask of this cidr"""
        ...
```

(continues on next page)

```
    ...
```

We would also like CIDRs to be validated by bravado-core whenever they are part of a HTTP request or response.

Create a `bravado_core.formatter.SwaggerFormat` to define the CIDR format:

```python
from bravado_core.formatter import SwaggerFormat


def validate_cidr(cidr_string):
    if '/' not in cidr_string:
        raise SwaggerValidationError('CIDR {0} is invalid'.format(cidr_string))

cidr_format = SwaggerFormat(
    # name of the format as used in the Swagger spec
    format='cidr',

    # Callable to convert a python CIDR object to a string
    to_wire=lambda cidr_object: cidr_object.cidr,

    # Callable to convert a string to a python CIDR object
    to_python=lambda cidr_string: CIDR(cidr_string),

    # Callable to validate the cidr in string form
    validate=validate_cidr
)
```

3.2 Configuring user-defined formats

Now that we have a `cidr_format`, just pass it to a `Spec` as part of the `config` parameter on `Spec` creation.

```python
from bravado_core.spec import Spec

spec_dict = json.loads(open('swagger.json', 'r').read())
config = {
    'validate_responses': True,
    'validate_requests': True,
    'formats': [cidr_format],
}
spec = Spec.from_dict(spec_dict, config=config)
```

All validation and processing of HTTP requests and responses will now use the configured format where appropriate.

3.3 Putting it all together

A simple example of passing a CIDR object to a request and getting a list of CIDR objects back from the response.

```json
{
    "paths": {
        "/get_overlapping_cidrs": {
            "get": {
                "parameters": [
```

```
                {
                    "name": "cidr",
                    "in": "query",
                    "type": "string",
                    "format": "cidr"
                }
            ],
            "responses": {
                "200": {
                    "description": "List of overlapping cidrs",
                    "schema": {
                        "type": "array",
                        "items": {
                            "type": "string",
                            "format": "cidr"
                        }
                    }
                }
            }
        }
    }
}
```

```python
from bravado_core.spec import Spec
from bravado_core.response import unmarshal_response
from bravado_core.param import marshal_param


# Retrieve the swagger spec from the server and json.load() it
spec_dict = ...

# Create cidr_format add it to the config dict
config = ...

# Create a bravado_core.spec.Spec
swagger_spec = Spec.from_dict(spec_dict, config=config)

# Get the operation to invoke
op = swagger_spec.get_op_for_request('GET', '/get_overlapping_cidrs')

# Get the Param that represents the cidr query parameter
cidr_param = op.params.get('cidr')

# Create a CIDR object - to_wire() will be called on this during marshalling
cidr_object = CIDR('192.168.1.1/24')
request_dict = {}

# Marshal the cidr_object into the request_dict.
marshal_param(cidr_param, cidr_object, request_dict)

# Lots of hand-wavey stuff here - use whatever http client you have to
# send the request and receive a response
response = http_client.send(request_dict)

# Extract the list of cidrs
cidrs = unmarshal_response(response)
```

```python
# Verify cidrs are CIDR objects and not strings
for cidr in cidrs:
    assert type(cidr) == CIDR
```

3.4 Overriding built-in formats is also possible with a user-defined format

By default format 'double' is internally converted to a float in python. This runs the risk of being imprecise due to the way floats are handled. If you would like to instead use the decimal.Decimal() type for more precision, you can override this built-in format. See Also: Floating Point Precision.

You'd define the type:

```python
from decimal import Decimal
import re

is_decimal = re.compile(r'^\d+(?:\.\d+)?$')
def validate_decimaltype(x):
  """Validate input is a str in valid decimal format"""
  if not (isinstance(x, str) and is_decimal.match(x)):
      raise bravado_core.exception.SwaggerValidationError()

mydouble = SwaggerFormat(
  format='double',
  to_wire=lambda x: str(x) if isinstance(x, Decimal) else str(Decimal(x)),
  to_python=lambda x: x if isinstance(x, Decimal) else Decimal(x),
  validate=validate_decimaltype,
  description="model format double internally as Decimal()"
)
```

Then in your config block you include this format:

```python
config = {
    'formats': [mydouble],
    ...
}

# Create a bravado_core.spec.Spec
swagger_spec = Spec.from_dict(spec_dict, config=config)
```

3.5 Note about using precise Decimal format in Spec

The above works when the openapi schema is written as `string(double)` e.g. the spec passes the value as string on the wire and format is double. If the spec said it was a number(double), it is likely that json will first convert the number from the wire to a float and then pass that into Decimal() with unguaranteed precision. The calls to json would need `use_decimals=True` for that to work.

Changelog

4.1 5.10.2 (2019-03-06)

- Fix issue with jsonschema 3.0 compatibility on Python 2.7 - Issue #318, PR #319, 320

4.2 5.10.1 (2019-01-15)

- Ensure that flattening Swagger Spec with `$ref:   None` will not cause unbounded recursion - PR #315
- Enhance `is_ref` detection to recognize only objects with `$ref` attribute and `string` value as reference - PR #315

4.3 5.10.0 (2018-11-20)

- Add `use_spec_url_for_base_path` configuration option - PR #300 - Thanks DStape for your contribution!
- Ensure `jsonschema >= 3` compatibility - PR #304
- Minor improvement on discriminator validation - PR #302

4.4 5.0.7 (2018-09-18)

- Fix security object validation issue - PR #294
- Fix unbounded recursion during object validation (if `internally_dereference_refs` is enabled) - PR #297
- Fix api_url generation - PR #295. Thanks mulmschneider for your contribution!

4.5 5.0.6 (2018-08-06)

- Swagger Spec flattening - fix regression that led to some model definitions not having the `x-model` marker anymore - PR #293
- Fix marshalling of array params that use collectionFormat `multi` - PR #292

4.6 5.0.5 (2018-08-02)

- Swagger Spec flattening - use model names instead of generated keys - PR #284
- Swagger Spec flattening - replace inline models with top level definitions - PR #285
- Fix query parameter marshalling in case of boolean parameters - Issue #281 - PR #286

4.7 5.0.4 (2018-06-29)

- Properly sanitize names with multiple leading digits, handle more edge cases correctly - PR #282

4.8 5.0.3 (2018-06-06)

- Make sure spaces in path param values are quoted using percent notation instead of using +. Issue #278, PR #279

4.9 5.0.2 (2018-06-04)

- Fix regression if *internally_dereference_refs* is used. Issue #275, PR #276

4.10 5.0.1 (2018-05-30)

- No longer make sure that all config keys are known; this allows users of the library to store additional configuration. - PR #274

4.11 5.0.0 (2018-05-30)

- Refactor: model discovery is now handled in `bravado_core.model` - PR #270
- Remove deprecated methods from Model type - PR #270
- Remove deprecated parameters from `bravado_core.spec_flattening.flattened_spec` - PR #269
- Ensure that models in *#/definitions* of referenced files are discovered - PR #273

> **Warning:** This release contains breaking changes! The signature of `bravado_core.spec_flattening.flattened_spec` has been updated. The following methods have been removed from the public interface: `bravado_core.model.tag_models`, `bravado_core.model.bless_models`, `bravado_core.model.collect_models` and `bravado_core.spec.post_process_spec`.

4.12 4.13.4 (2018-05-24)

- Fix marsharling and unmarshaling of optional body parameters. PR #268

4.13 4.13.3 (2018-05-16)

- Add support for Content-Disposition filename - PR #262. Thanks elmirjagudin for your contribution!
- Improve specs flattening and dereferencing in case of relative references - PR #263

4.14 4.13.2 (2018-03-19)

- Fix bug where multiple schemes in the spec would sometimes cause a `SwaggerSchemaError` - PR #260

4.15 4.13.1 (2018-03-02)

- Catch TypeErrors during param unmarshalling, allowing JSON Schema to handle the error - Issue #258, PR #259. Thanks Nick DiRienzo for your contribution!

4.16 4.13.0 (2018-02-23)

- Models are generated only for objects - PR #246.
- Fix: ensure that models do not have references if `internally_dereference_refs` is used - PR #247.
- Model name detection uses title attribute too - PR #249.
- Duplicated models do not raise exception if `use_models` is not used - PR #253.
- Alert or warn if pre-tagged duplicate models are found - PR #254.

4.17 4.12.1 (2018-02-07)

- Make sure unsanitized param names are used when unmarshalling a request - PR #245.
- Expose the determine_object_type method as part of our API - PR #244.

4.18 4.12.0 (2018-02-06)

- Sanitize resource and parameter names so that they're valid Python identifiers. It uses the same logic as for operationIds - invalid characters are replaced with underscores, multiple consecutive underscores are merged into one, and leading / trailing underscores are removed. Using the unsanitized names will still work - Issue #200, PR #243.
- Allow overriding built-in default formats - Issue #235, PR #240. Thanks Brian J. Dowling for your contribution!
- Include additionalProperties in a models' __repr__ - PR #242. Thanks again Brian J. Dowling!

4.19 4.11.5 (2018-01-30)

- Use yaml.safe_load for parsing specs - PR #241.

4.20 4.11.4 (2018-01-19)

- Properly quote request parameters sent as part of the URL path - PR #237, #238.

4.21 4.11.3 (2018-01-16)

- Remove strict isinstance check when marshalling models - PR #236.

4.22 4.11.2 (2018-01-08)

- Ensure `internally_dereference_refs` works with recursive specs - PR #234.

4.23 4.11.1 (2017-12-18)

- Speed up marshalling and unmarshalling of objects - PR #226.
- Use msgpack-python instead of u-msgpack-python for performance improvements - Issue #227, PR #228.

4.24 4.11.0 (2017-11-09)

- Add support for msgpack in responses (i.e. when unmarshalling) - Issue #214, PR #216.
- Improve performance by removing debug logging when dereferencing - PR #208.

4.25 4.10.1 (2017-11-06)

- Don't remove unrecognized configs; fixes compatibility with bravado - PR #218.

4.26 4.10.0 (2017-11-03)

- New config `internally_dereference_refs` that can significantly speed up unmarshalling. Currently disabled by default - PR #204.
- Added support for new extension `x-sensitive` to scrub sensitive values from validation errors. Please check the documentation for further details - PR #213.
- Fixed an issue that would cause validation errors if `obj_type` was `None` - PR #196.
- Fixed handling of defaults for array parameters - PR #199.
- Performance improvements - PR #207.

4.27 4.9.1 (2017-09-19)

- Properly marshal a model even if it's not created from the same `Spec` instance - PR #194.

4.28 4.9.0 (2017-09-11)

- `type` is no longer required. By default, validation will not be performed if `type` is omitted. This is configurable with `default_type_to_object` - Issue #166, #192, PR #183, #193

4.29 4.8.4 (2017-09-06)

- Make sure all models are properly tagged when flattening the spec - PR #191.

4.30 4.8.3 (2017-09-05)

- Improve spec flattening: recognize response objects and expose un-referenced models - PR #184.
- Fix a bug when marshalling properties with no spec that have the value `None` - PR #189.

4.31 4.8.2 (2017-09-04)

- Fix marshalling of `null` values for properties with `x-nullable` set to `true` - Issue #185, PR #186. Thanks Jan Baraniewski for the contribution!
- Add `_asdict()` method to each model, similar to what namedtuples have - PR #188.

4.32 4.8.1 (2017-08-24)

- Make unmarshalling objects roughly 30% faster - PR #182.

4.33 4.8.0 (2017-07-15)

- Add support for Swagger spec flattening - PR #177.
- Fix handling of API calls that return non-JSON content (specifically text content) - PR #175. Thanks mostrows2 for your contribution!
- Fix error message text when trying to unmarshal an invalid model - PR #179.

4.34 4.7.3 (2017-05-05)

- Fix support for object composition (allOf) for data passed in the request body - PR #167. Thanks Zi Li for your contribution!
- Return the default value for an optional field missing in the response - PR #171.

4.35 4.7.2 (2017-03-23)

- Fix unmarshalling of null values for properties with no spec - Issue #163, PR #165.

4.36 4.7.1 (2017-03-22)

- Fix backward-incompatible Model API change which renames all model methods to have a single underscore infront of them. A deprecation warning has been added - Issue #160, PR #161. Thanks Adam Ever-Hadani for the contribution!

4.37 4.7.0 (2017-03-21)

- Added support for nullable fields in the format validator - PR #143. Thanks Adam Ever-Hadani
- Add include_missing_properties configuration - PR #152
- Consider default when unmarshalling - PR #154
- Add discriminator support - PR #128, #159. Thanks Michael Jared Lumpe for your contribution
- Make sure pre-commit hooks are installed and run when running tests - PR #155, #158

4.38 4.6.1 (2017-02-15)

- Fix unmarshalling empty array types - PR #148
- Removed support for Python 2.6 - PR #147

4.39 4.6.0 (2016-11-28)

- Security Requirement validation (for ApiKey) - PR #124
- Allow self as name for model property, adds new "create" alternate model constructor - Issue #125, PR #126.
- Allow overriding of security specs - PR #121
- Adds minimal support for responses with text/* content_type.

4.40 4.5.1 (2016-09-27)

- Add marshal and unmarshal methods to models - PR #113, #120.

4.41 4.5.0 (2016-09-12)

- Support for model composition through the allOf property - Issue #7, PR #63, #110. Thanks David Bartle for the initial contribution!
- Fix issue with header parameter values being non-string types - PR #115.

4.42 4.4.0 (2016-08-26)

- Adds support for security scheme definitions, mostly focusing on the "apiKey" type - PR #112.

4.43 4.3.2 (2016-08-17)

- Fixes around unmarshalling, x-nullable and required behavior - Issue #108, PR #109. Big thanks to Zachary Roadhouse for the report and pull request!
- Fix AttributeError when trying to unmarshal a required array param that's not present - PR #111.

4.44 4.3.1 (2016-08-09)

- Check if a parameter is bool-type before assuming it's a string - PR #107. Thanks to Nick DiRienzo for the pull request!

4.45 4.3.0 (2016-08-04)

- Add support for `x-nullable` - Issue #47, PR #64 and #103. Thanks to Andreas Hug for the pull request!
- Fix support for vendor extensions at the path level - PR #95, #106. Thanks to Mikołaj Siedlarek for the initial pull request!

4.46 4.2.5 (2016-07-27)

- Add basepython python2.7 for flake8, docs, and coverage tox commands

4.47 4.2.4 (2016-07-26)

- coverage v4.2 was incompatible and was breaking the build. Added –append for the fix.

4.48 4.2.3 (2016-07-26)

- Accept tuples as a type list as well.

4.49 4.2.2 (2016-04-01)

- Fix marshalling of an optional array query parameter when not passed in the service call - PR #87

4.50 4.2.1 (2016-03-23)

- Fix optional enums in request params - Issue #77
- Fix resolving refs during validation - Issue #82

4.51 4.2.0 (2016-03-10)

- More robust handling of operationId which contains non-standard chars - PR #76
- Provide a client ingestible version of spec_dict with x-scope metadata removed. Accessible as Spec.client_spec_dict - Issue #78

4.52 4.1.0 (2016-03-01)

- Better handling of query parameters that don't have a value - Issue #68
- Allow marshalling of objects which are subclasses of dict - PR #61
- Fix boolean query params to support case-insensetive true/false and 0/1 - Issue #70
- Support for Swagger specs in yaml format - Issue #42
- Fix validation of server side request parameters when collectionFormat=multi and item type is not string - Issue #66
- Fix unmarshaling of server side request parameters when collectionFormat=multi and cardinality is one - PR #75

4.53 4.0.1 (2016-01-11)

- Fix unmarshalling of an optional array query parameter when not passed in the query string.

4.54 4.0.0 (2015-11-17)

- Support for recursive $refs - Issue #35

- Requires swagger-spec-validator 2.0.1

- Unqualified $refs no longer supported. Bad: `{"$ref": "User"}` Good: `{"$ref": "#/definitions/User"}`

- Automatic tagging of models is only supported in the root swagger spec file. If you have models defined in $ref targets that are in other files, you must manually tag them with 'x-model' for them to be available as python types. See Model Discovery for more info.

4.55 3.1.1 (2015-10-19)

- Fix the creation of operations that contain shared parameters for a given endpoint.

4.56 3.1.0 (2015-10-19)

- Added http `headers` to `bravado_core.response.IncomingResponse`.

4.57 3.0.2 (2015-10-12)

- Added docs on how to use user-defined formats.

- Added docs on how to configure bravado-core.

- *formats* added as a config option

4.58 3.0.1 (2015-10-09)

- Automatically tag models in external $refs - Issue #45 - see Model Discovery for more info.

4.59 3.0.0 (2015-10-07)

- User-defined formats are now scoped to a Swagger spec - Issue #50 (this is a non-backwards compatible change)

- Deprecated bravado_core.request.RequestLike and renamed to bravado_core.request.IncomingRequest

- Added *make docs* target and updated docs (still needs a lot of work though)

4.60 2.4.1 (2015-09-30)

- Fixed validation of user-defined formats - Issue #48

4.61 2.4.0 (2015-08-13)

- Support relative '$ref' external references in swagger.json
- Fix dereferencing of jsonref when given in a list

4.62 2.3.0 (2015-08-10)

- Raise MatchingResponseNotFound instead of SwaggerMappingError when a response can't be matched to the Swagger schema.

4.63 2.2.0 (2015-08-06)

- Add reason to IncomingResponse

4.64 2.1.0 (2015-07-17)

- Handle user defined formats for serialization and validation.

4.65 2.0.0 (2015-07-13)

- Move http invocation to bravado
- Fix unicode in model docstrings
- Require swagger-spec-validator 1.0.12 to pick up bug fixes

4.66 1.1.0 (2015-06-25)

- Better unicode support
- Python 3 support

4.67 1.0.0-rc2 (2015-06-01)

- Fixed file uploads when marshaling a request
- Renamed ResponseLike to IncomingResponse
- Fixed repr of a model when it has an attr with a unicode value

4.68 1.0.0-rc1 (2015-05-26)

- Use basePath when matching an operation to a request
- Refactored exception hierarchy
- Added use_models config option

4.69 0.1.0 (2015-05-13)

- Initial release

CHAPTER 5

Indices and tables

- genindex
- modindex
- search