

---

**bravado**

**Jun 20, 2018**



---

## Contents

---

<b>1</b>	<b>Quickstart</b>	<b>3</b>
<b>2</b>	<b>Configuration</b>	<b>7</b>
<b>3</b>	<b>Making requests with bravado</b>	<b>11</b>
<b>4</b>	<b>Advanced Usage</b>	<b>13</b>
<b>5</b>	<b>Testing code that uses bravado</b>	<b>19</b>
<b>6</b>	<b>API reference</b>	<b>23</b>
<b>7</b>	<b>Changelog</b>	<b>39</b>
<b>8</b>	<b>Indices and tables</b>	<b>45</b>
	<b>Python Module Index</b>	<b>47</b>



Bravado is a python client library for Swagger 2.0 services.

More information on Swagger can be found [on the Swagger website](#)

It aims to be a complete replacement to [swagger codegen](#).

Features include:

- Dynamically generated client - no code generation needed!
- [Synchronous](#) and [Asynchronous](#) http clients out of the box.
- Strict validations to verify that your Swagger Schema is [v2.0](#) compatible.
- HTTP request and response validation against your Swagger Schema.
- Swagger models as Python types (no need to deal with JSON).
- REPL friendly navigation of your Swagger schema with docstrings for Resources, Operations and Models.
- Ingestion of your Swagger schema via http or a local file path.

Contents:



### 1.1 Usage

Install the latest stable version from PyPi:

```
$ pip install --upgrade bravado
```

### 1.2 Your first Hello World! (or Hello Pet)

Here is a simple example to try from a REPL (like IPython):

```
from bravado.client import SwaggerClient

client = SwaggerClient.from_url("http://petstore.swagger.io/v2/swagger.json")
pet = client.pet.getPetById(petId=42).response().result
```

If you were lucky, and pet Id with 42 was present, you will get back a result. It will be a dynamically created instance of `bravado.model.Pet` with attributes `category`, etc. You can even try `pet.category.id` or `pet.tags[0]`.

Sample Response:

```
Pet(category=Category(id=0L, name=u''), status=u'', name=u'', tags=[Tag(id=0L, name=u'
↪')], photoUrls=[u''], id=2)
```

If you got a 404, try some other `petId`.

## 1.3 Lets try a POST call

Here we will demonstrate how bravado hides all the JSON handling from the user, and makes the code more Pythonic.

```
Pet = client.get_model('Pet')
Category = client.get_model('Category')
pet = Pet(id=42, name="tommy", category=Category(id=24))
client.pet.addPet(body=pet).response().result
```

## 1.4 Time to get Twisted! (Asynchronous client)

bravado provides an out of the box asynchronous http client with an optional timeout parameter.

*Your first Hello World! (or Hello Pet)* above can be rewritten to use the asynchronous Fido client like so:

```
from bravado.client import SwaggerClient
from bravado.fido_client import FidoClient

client = SwaggerClient.from_url(
    'http://petstore.swagger.io/v2/swagger.json',
    FidoClient()
)

result = client.pet.getPetById(petId=42).result(timeout=4)
```

---

**Note:** timeout parameter here is the timeout (in seconds) the call will block waiting for the complete response. The default timeout is to wait indefinitely.

---

---

**Note:** To use Fido client you should install bravado with fido extra via `pip install bravado[fido]`.

---

## 1.5 This is too fancy for me! I want a simple dict response!

bravado has taken care of that as well. Configure the client to not use models.

```
from bravado.client import SwaggerClient
from bravado.fido_client import FidoClient

client = SwaggerClient.from_url(
    'http://petstore.swagger.io/v2/swagger.json',
    config={'use_models': False}
)

result = client.pet.getPetById(petId=42).result(timeout=4)
```

result will look something like:



```
{
  'category': {
    'id': 0L,
    'name': u''
  },
  'id': 2,
  'name': u'',
  'photoUrls': [u''],
  'status': u'',
  'tags': [
    {'id': 0L, 'name': u''}
  ]
}
```



### 2.1 Client Configuration

You can configure certain behaviours when creating a `SwaggerClient`.

`bravado` and `bravado-core` use the same config dict. The full documentation for [bravado-core config keys](#) is available too.

```
from bravado.client import SwaggerClient, SwaggerFormat

my_super_duper_format = SwaggerFormat(...)

config = {
    # === bravado config ===

    # What class to use for response metadata
    'response_metadata_class': 'bravado.response.BravadoResponseMetadata',

    # Do not use fallback results even if they're provided
    'disable_fallback_results': False,

    # DEPRECATED: Determines what is returned by HttpFuture.result().
    # Please use HttpFuture.response() for accessing the http response.
    'also_return_response': False,

    # === bravado-core config ===

    # Validate incoming responses
    'validate_responses': True,

    # Validate outgoing requests
    'validate_requests': True,

    # Validate the swagger spec
```

(continues on next page)

(continued from previous page)

```

'validate_swagger_spec': True,

# Use models (Python classes) instead of dicts for #/definitions/{models}
'use_models': True,

# List of user-defined formats
'formats': [my_super_duper_format],
}

client = SwaggerClient.from_url(..., config=config)

```

Config key	Type	Description
<i>response_metadata_class</i>	string	<p>The Metadata class to use; see <i>Custom response metadata</i> for details.</p> <p>Default: <code>bravado.response.BravadoResponseMetadata</code></p>
<i>disable_fallback_results</i>	boolean	<p>Whether to disable returning fallback results, even if they're provided as an argument to <code>HttpFuture.response()</code>.</p> <p>Default: False</p>
<i>also_return_response</i>	boolean	<p>Determines what is returned by the service call.</p> <p>Specifically, the return value of <code>HttpFuture.result()</code>.</p> <p>When False, the swagger result is returned.</p> <p>When True, the tuple (swagger result, http response) is returned. Has no effect on the return value of <code>HttpFuture.response()</code>.</p> <p>See <i>Getting access to the HTTP response</i>.</p> <p>Default: False</p>

## 2.2 Per-request Configuration

Configuration can also be applied on a per-request basis by passing in `_request_options` to the service call.

```
client = SwaggerClient.from_url(...)
request_options = { ... }
client.pet.getPetById(petId=42, _request_options=request_options).response().result
```

Config key	Type	Default	Description
<i>connect_timeout</i>	float	N/A	TCP connect timeout in seconds. This is passed along to the <code>http_client</code> when making a service call.
<i>headers</i>	dict	N/A	Dict of http headers to send with the outgoing request.
<i>response_callbacks</i>	list of callables	<code>[]</code>	<p>List of callables that are invoked after the incoming response has been validated and unmarshalled but before being returned to the calling client. This is useful for client decorators that would like to hook into the post-receive event. The callables are executed in the order they appear in the list.</p> <p>Two parameters are passed to each callable:</p> <ul style="list-style-type: none"> <li>- <code>incoming_response</code> of type <code>bravado_core.response.IncomingResponse</code></li> <li>- <code>operation</code> of type <code>bravado_core.operation.Operation</code></li> </ul>
<i>timeout</i>	float	N/A	TCP idle timeout in seconds. This is passed along to the <code>http_client</code> when making a service call.
<b>10</b> <i>use_msgpack</i>	boolean	False	<p><b>Chapter 2. Configuration</b></p> <p>If a msgpack serialization is desired for the response. This</p>

---

### Making requests with bravado

---

When you call `SwaggerClient.from_url()` or `SwaggerClient.from_spec()`, Bravado takes a Swagger (OpenAPI) 2.0 spec and returns a `SwaggerClient` instance that you can use to make calls to the service described in the spec. You make calls by doing Python method calls in the form of `client.resource.operation(operation_params)`. Use `dir(client)` to see all available resources.

#### 3.1 Resources and operations

Resources are generated for each tag that exists in your Swagger spec. If an operation has no tags then the left-most element of its path is taken as resource name. So in the case of an operation with the path `/pet/find`, `pet` will be the resource.

The operation name will be the (*sanitized*) `operationId` value from the Swagger spec. If there is no `operationId`, it will be generated. We highly recommend providing operation IDs for all operations. Use `dir(client.resource)` to see a list of all available operations.

The operation method expects keyword arguments that have the same (*sanitized*) names as in the Swagger spec. Use corresponding Python types for the values - if the Swagger spec says a parameter is of type `boolean`, provide it as a Python `bool`.

#### 3.2 Futures and responses

The return value of the operation method is a `HttpFuture`. To access the response, call `HttpFuture.response()`. This call will block, i.e. it will wait until the response is received or the timeout you specified is reached.

If the request succeeded and the server returned a HTTP status code between 100 and 299, the return value of `HttpFuture.response()` will be a `BravadoResponse` instance. You may access the Swagger result of your call through `BravadoResponse.result`.

If the server sent a response with a HTTP code of 400 or higher, by default a subclass of `HTTPError` will be raised when you call `HttpFuture.response()`. The exception gives you access to the Swagger result (`HTTPError.swagger_result`) as well as the HTTP response object (`HTTPError.response`).

### 3.3 Response metadata

`BravadoResponse.metadata` is an instance of `BravadoResponseMetadata` that provides you with access to the HTTP response including headers and HTTP status code, request timings and whether a fallback result was used (see *Working with fallback results*).

You're able to provide your own implementation of `BravadoResponseMetadata`; see *Custom response metadata* for details.

### 3.4 Sanitizing names

Not all characters that the Swagger spec allows for names are valid Python identifiers. In particular, spaces and the `-` character can be troublesome. bravado sanitizes resource, operation and parameter names according to these rules:

- Any character that is not a letter or number is converted to an underscore (`_`)
- Collapse multiple consecutive underscores to one
- Remove leading and trailing underscores
- Remove leading numbers



### 4.1 Validations

bravado validates the schema against the Swagger 2.0 Spec. Validations are also done on the requests and the responses.

Validation example:

```
pet = Pet(id="I should be integer :", name="tommy")
client.pet.addPet(body=pet).response().result
```

will result in an error like so:

```
TypeError: id's value: 'I should be integer :(' should be in types (<type 'long'>,
↪<type 'int'>)
```

---

**Note:** If you'd like to disable validation of outgoing requests, you can set `validate_requests` to `False` in the config passed to `SwaggerClient.from_url(...)`.

The same holds true for incoming responses with the `validate_responses` config option.

---

### 4.2 Adding Request Headers

bravado allows you to pass request headers along with any request.

```
Pet = client.get_model('Pet')
Category = client.get_model('Category')
pet = Pet(id=42, name="tommy", category=Category(id=24))
swagger_client.pet.addPet(
    body=pet,
```

(continues on next page)

(continued from previous page)

```
_request_options={"headers": {"foo": "bar"}},
).response().result
```

## 4.3 Docstrings

bravado provides docstrings to operations and models to quickly get the parameter and response types. Due to an implementation limitation, an operation's docstring looks like a class docstring instead of a function docstring. However, the most useful information about parameters and return type is present in the Docstring section.

**Note:** The help built-in does not work as expected for docstrings. Use the ? method instead.

```
>> petstore.pet.getPetById?

Type:          CallableOperation
String Form:<bravado.client.CallableOperation object at 0x241b5d0>
File:          /some/dir/bravado/bravado/client.py
Definition: c.pet.getPetById(self, **op_kwargs)
Docstring:
[GET] Find pet by ID

Returns a single pet

:param petId: ID of pet to return
:type petId: integer
:returns: 200: successful operation
:rtype: object
:returns: 400: Invalid ID supplied
:returns: 404: Pet not found
Constructor Docstring::type operation: :class:`bravado_core.operation.Operation`
Call def:      c.pet.getPetById(self, **op_kwargs)
Call docstring:
Invoke the actual HTTP request and return a future that encapsulates
the HTTP response.

:rtype: :class:`bravado.http_future.HTTPFuture`
```

Docstrings for models can be retrieved as expected:

```
>> pet_model = petstore.get_model('Pet')
>> pet_model?

Type:          type
String Form:<class 'bravado_core.model.Pet'>
File:          /some/dir/bravado_core/model.py
Docstring:
Attributes:

category: Category
id: integer
name: string
photoUrls: list of string
status: string - pet status in the store
```

(continues on next page)

(continued from previous page)

```
tags: list of Tag
Constructor information:
Definition: pet_type(self, **kwargs)
```

## 4.4 Default Values

bravado uses the default values from the spec if the value is not provided in the request.

In the [Pet Store](#) example, operation `findPetsByStatus` has a default of `available`. That means, bravado will plug that value in if no value is provided for the parameter.

```
client.pet.findPetByStatus()
```

## 4.5 Loading swagger.json by file path

bravado also accepts `swagger.json` from a file path. Like so:

```
client = SwaggerClient.from_url('file:///some/path/swagger.json')
```

Alternatively, you can also use the `load_file` helper method.

```
from bravado.swagger_model import load_file

client = SwaggerClient.from_spec(load_file('/path/to/swagger.json'))
```

## 4.6 Getting access to the HTTP response

The default behavior for a service call is to return the swagger result like so:

```
pet = client.pet.getPetById(petId=42).response().result
print pet.name
```

However, there are times when it is necessary to have access to the actual HTTP response so that the HTTP headers or HTTP status code can be used. Simply save the response object (which is a [BravadoResponse](#)) and use its `incoming_response` attribute to access the incoming response:

```
petstore = SwaggerClient.from_url(
    'http://petstore.swagger.io/swagger.json',
    config={'also_return_response': True},
)
pet_response = petstore.pet.getPetById(petId=42).response()
http_response = pet_response.incoming_response
assert isinstance(http_response, bravado_core.response.IncomingResponse)
print http_response.headers
print http_response.status_code
print pet.name
```

## 4.7 Working with fallback results

By default, if the server returns an error or doesn't respond in time, you have to catch and handle the resulting exception accordingly. A simpler way would be to use the support for fallback results provided by `HttpFuture.response()`.

`HttpFuture.response()` takes an optional argument `fallback_result` which is a callable that returns a Swagger result. The callable takes one mandatory argument: the exception that would have been raised normally. This allows you to return different results based on the type of error (e.g. a `BravadoTimeoutError`) or, if a server response was received, on any data pertaining to that response, like the HTTP status code.

In the simplest case, you can just specify what you're going to return:

```
petstore = SwaggerClient.from_url('http://petstore.swagger.io/swagger.json')
response = petstore.pet.findPetsByStatus(status=['available']).response(
    timeout=0.5,
    fallback_result=lambda e: [],
)
```

This code will return an empty list in case the server doesn't respond quickly enough (or it responded quickly enough, but returned an error).

### 4.7.1 Customizing which error types to handle

By default, the fallback result will be used either when the server doesn't send the response in time or when it returns a server error (i.e. a result with a HTTP 5XX status code). To override this behavior, specify the `exceptions_to_catch` argument to `HttpFuture.response()`.

The default is defined in `bravado.http_future.FALLBACK_EXCEPTIONS`. See [bravado.exception](#) for a list of possible exception types.

### 4.7.2 Models and fallback results

But what if you're using models (the default) and the endpoint you're calling returns one? You'll have to return one as well from your `fallback_result` function to stay compatible with the rest of your code:

```
petstore = SwaggerClient.from_url('http://petstore.swagger.io/swagger.json')
response = petstore.pet.getPetById(petId=101).response(
    timeout=0.5,
    fallback_result=lambda e: petstore.get_model('Pet')(name='No Pet found',
    ↳photoUrls=[]),
)
```

Two things to note here: first, use `SwaggerClient.get_model()` to get the model class for a model name. Second, since `name` and `photoUrls` are required fields for this model, we probably should not leave them empty (if we do they'd still be accessible, but the value would be `None`). It's up to you how you decide to deal with this case.

`BravadoResponseMetadata.is_fallback_result` will be `True` if a fallback result has been returned by the call to `HttpFuture.response()`.

### 4.7.3 Testing fallback results

You can trigger returning fallback results for testing purposes. Just set the option `force_fallback_result` to `True` in the request configuration (see [Per-request Configuration](#)). In this case a `ForcedFallbackResultError`

exception will be passed to your fallback result callback, so make sure you handle it properly.

## 4.8 Custom response metadata

Sometimes, there's additional metadata in the response that you'd like to make available easily. This case arises most often if you're using bravado to talk to internal services. Maybe you have special HTTP headers that indicate whether a circuit breaker was triggered? bravado allows you to customize the metadata and provide custom attributes and methods.

In your code, create a class that subclasses `bravado.response.BravadoResponseMetadata`. In the implementation of your properties, use `BravadoResponseMetadata.headers` to access response headers, or `BravadoResponseMetadata.incoming_response` to access any other part of the HTTP response.

If, for some reason, you need your own `__init__` method, make sure that your signature accepts any positional and keyword argument, and that you call the base method with these arguments from your own implementation. That way, your class will remain compatible with the base class even if new arguments get added to the `__init__` method. Example minimal implementation:

```
class MyResponseMetadata(ResponseMetadata):
    def __init__(self, *args, **kwargs):
        super(MyResponseMetadata, self).__init__(*args, **kwargs)
```

While developing custom `BravadoResponseMetadata` classes we recommend to avoid, if possible, the usage of attributes for data that's expensive to compute. Since the object will be created for every response, implementing these fields as properties makes sure the evaluation is only done if the field is accessed.



---

## Testing code that uses bravado

---

Writing tests is crucial in making sure your code works and behaves as expected. bravado ships with two classes that will help you create robust unit tests to verify the correctness of your code. We'll be using the excellent `pytest` library in this example.

First of all, let's define the code we'd like to test:

```
from itertools import chain

from bravado.client import SwaggerClient

def get_available_pet_photos():
    petstore = SwaggerClient.from_url(
        'http://petstore.swagger.io/swagger.json',
    )
    pets = petstore.pet.findPetsByStatus(status=['available']).response(
        timeout=0.5,
        fallback_result=lambda e: [],
    ).result

    return chain.from_iterable(pet.photoUrls for pet in pets)
```

First of all, in order to make sure your code doesn't do any network requests, you need to mock out the bravado client:

```
import mock
import pytest

from bravado.client import SwaggerClient

@pytest.fixture
def mock_client():
    mock_client = mock.Mock(name='mock SwaggerClient')
    with mock.patch.object(SwaggerClient, 'from_url', return_value=mock_client):
        yield mock_client
```

Now we can mock out that call to `findPetsByStatus` by using the `bravado.testing.response_mock.BravadoResponseMock` class:

```
import mock

from bravado.testing.response_mock import BravadoResponseMock

from mypackage import get_available_pet_photos

def test_get_available_pet_photos(mock_client):
    mock_pets = [
        mock.Mock(
            photoUrls=['https://example.com/image.png'],
        ),
        mock.Mock(
            photoUrls=[
                'https://example.com/image2.png',
                'https://example.com/image3.png',
            ],
        ),
    ]

    mock_client.pet.findPetsByStatus.return_value.response = BravadoResponseMock(
        result=mock_pets,
    )

    pet_photos = get_available_pet_photos()

    assert list(pet_photos) == [
        'https://example.com/image.png',
        'https://example.com/image2.png',
        'https://example.com/image3.png',
    ]
```

Note that it's your responsibility to ensure that what you set as result for `BravadoResponseMock` is sufficiently similar to what bravado would return in production. We've used a `Mock` class here; another option is to define namedtuples that correspond to your Swagger spec objects. This gives you even greater confidence in the correctness of your code since access to undefined fields will result in an error.

## 5.1 Testing degraded responses

Use `FallbackResultBravadoResponseMock` to test *fallback results*. It works similarly, but you don't have to pass the result to the constructor, since your fallback result callback will determine the result. Let's add another test to verify our fallback result code path works properly:

```
from bravado.testing.response_mock import FallbackResultBravadoResponseMock

from example import get_available_pet_photos

def test_get_available_pet_photos_fallback_result(mock_client):
    mock_client.pet.findPetsByStatus.return_value\
        .response = FallbackResultBravadoResponseMock()

    pet_photos = get_available_pet_photos()

    assert list(pet_photos) == []
```



Note that you can pass in a custom exception instance to *FallbackResultBravadoResponseMock* if you need to trigger specific exception handling in your fallback result callback.

## 5.2 Setting custom response metadata

Both *BravadoResponseMock* as well as *FallbackResultBravadoResponseMock* accept an optional `metadata` argument. Just pass in an instance of *BravadoResponseMetadata* that you'd like to be used. A default one will be provided otherwise.



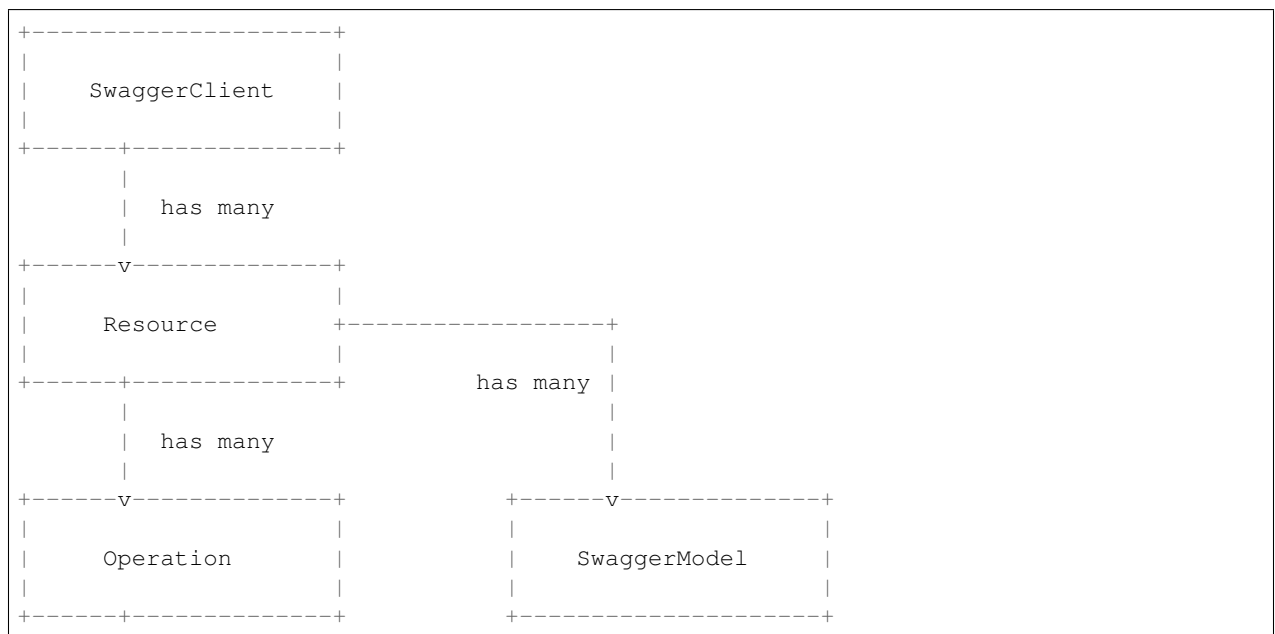
### 6.1 bravado Package

#### 6.1.1 bravado Package

#### 6.1.2 client Module

The *SwaggerClient* provides an interface for making API calls based on a swagger spec, and returns responses of python objects which build from the API response.

Structure Diagram:



(continues on next page)



To get a client

```
client = bravado.client.SwaggerClient.from_url(swagger_spec_url)
```

**class** `bravado.client.CallableOperation` (*operation, also\_return\_response=False*)  
 Bases: `object`

**class** `bravado.client.ResourceDecorator` (*resource, also\_return\_response=False*)  
 Bases: `object`

Wraps `bravado_core.resource.Resource` so that accesses to contained operations can be instrumented.

**class** `bravado.client.SwaggerClient` (*swagger\_spec, also\_return\_response=False*)  
 Bases: `object`

A client for accessing a Swagger-documented RESTful service.

**classmethod** `from_spec` (*spec\_dict, origin\_url=None, http\_client=None, config=None*)  
 Build a `SwaggerClient` from a Swagger spec in dict form.

#### Parameters

- **spec\_dict** – a dict with a Swagger spec in json-like form
- **origin\_url** (*str*) – the url used to retrieve the spec\_dict
- **config** – Configuration dict - see `spec.CONFIG_DEFAULTS`

**Return type** `bravado_core.spec.Spec`

**classmethod** `from_url` (*spec\_url, http\_client=None, request\_headers=None, config=None*)  
 Build a `SwaggerClient` from a url to the Swagger specification for a RESTful API.

#### Parameters

- **spec\_url** (*str*) – url pointing at the swagger API specification
- **http\_client** (`bravado.http_client.HttpClient`) – an HTTP client used to perform requests
- **request\_headers** (*dict*) – Headers to pass with http requests
- **config** – Config dict for bravado and bravado\_core. See `CONFIG_DEFAULTS` in **module: 'bravado\_core.spec'**. See `CONFIG_DEFAULTS` in **module: 'bravado.client'**.

**Return type** `bravado_core.spec.Spec`

**get\_model** (*model\_name*)

**bravado.client.construct\_params** (*operation, request, op\_kwargs*)

Given the parameters passed to the operation invocation, validates and marshals the parameters into the provided request dict.

**Parameters** `op_kwargs` – the kwargs passed to the operation invocation

**Raises** `SwaggerMappingError` on extra parameters or when a required parameter is not supplied.

`bravado.client.construct_request` (*operation, request\_options, \*\*op\_kwargs*)

Construct the outgoing request dict.

**Parameters**

- **request\_options** – `_request_options` passed into the operation invocation.
- **op\_kwargs** – parameter name/value pairs to be passed to the invocation of the operation.

**Returns** request in dict form

`bravado.client.inject_headers_for_remote_refs` (*request\_callable, request\_headers*)

Inject `request_headers` only when the request is to retrieve the remote refs in the swagger spec (vs being a request for a service call).

**Parameters**

- **request\_callable** – method on `http_client` to make a http request
- **request\_headers** – headers to inject when retrieving remote refs

### 6.1.3 config Module

`class bravado.config.BravadoConfig`

Bases: `bravado.config.BravadoConfig`

**static** `from_config_dict` (*config*)

`class bravado.config.RequestConfig` (*request\_options, also\_return\_response\_default*)

Bases: `object`

**additional\_properties** = {}

**also\_return\_response** = False

**connect\_timeout** = None

**force\_fallback\_result** = False

**headers** = {}

**response\_callbacks** = []

**timeout** = None

**use\_msgpack** = False

### 6.1.4 requests\_client Module

`class bravado.requests_client.ApiKeyAuthenticator` (*host, api\_key,*  
*param\_name=u'api\_key',*  
*param\_in=u'query'*)

Bases: `bravado.requests_client.Authenticator`

?api\_key authenticator.

This authenticator adds an API key via query parameter or header.

**Parameters**

- **host** – Host to authenticate for.
- **api\_key** – API key.
- **param\_name** – Query parameter specifying the API key.
- **param\_in** – How to send the API key. Can be ‘query’ or ‘header’.

**apply** (*request*)

Apply authentication to a request.

**Parameters** **request** – Request to add authentication information to.

**class** bravado.requests\_client.**Authenticator** (*host*)

Bases: `object`

Authenticates requests.

**Parameters** **host** – Host to authenticate for.

**apply** (*request*)

Apply authentication to a request.

**Parameters** **request** – Request to add authentication information to.

**matches** (*url*)

Returns true if this authenticator applies to the given url.

**Parameters** **url** – URL to check.

**Returns** True if matches host, port and scheme, False otherwise.

**class** bravado.requests\_client.**BasicAuthenticator** (*host, username, password*)

Bases: `bravado.requests_client.Authenticator`

HTTP Basic authenticator.

**Parameters**

- **host** – Host to authenticate for.
- **username** – Username.
- **password** – Password

**apply** (*request*)

Apply authentication to a request.

**Parameters** **request** – Request to add authentication information to.

**class** bravado.requests\_client.**RequestsClient**

Bases: `bravado.http_client.HttpClient`

Synchronous HTTP client implementation.

**apply\_authentication** (*request*)

**authenticated\_request** (*request\_params*)

**request** (*request\_params, operation=None, request\_config=None*)

**Parameters**

- **request\_params** (*dict*) – complete request data.
- **operation** (`bravado_core.operation.Operation`) – operation that this http request is for. Defaults to None - in which case, we’re obviously just retrieving a Swagger Spec.

- **request\_config** (`RequestConfig`) – per-request configuration

**Returns** HTTP Future object

**Return type**

`class bravado_core.http_future.HttpFuture`

**static separate\_params** (`request_params`)

Splits the passed in dict of request\_params into two buckets.

- `sanitized_params` are valid kwargs for constructing a `requests.Request(..)`
- `misc_options` are things like timeouts which can't be communicated to the Requests library via the `requests.Request(...)` constructor.

**Parameters** `request_params` – kitchen sink of request params. Treated as a read-only dict.

**Returns** tuple(`sanitized_params`, `misc_options`)

**set\_api\_key** (`host`, `api_key`, `param_name=u'api_key'`, `param_in=u'query'`)

**set\_basic\_auth** (`host`, `username`, `password`)

**class** `bravado.requests_client.RequestsFutureAdapter` (`session`, `request`, `misc_options`)

Bases: `bravado.http_future.FutureAdapter`

Mimics a `concurrent.futures.Future` for the purposes of making HTTP calls with the Requests library in a future-y sort of way.

**build\_timeout** (`result_timeout`)

Build the appropriate timeout object to pass to `session.send(...)` based on `connect_timeout`, the timeout passed to the service call, and the timeout passed to the result call.

**Parameters** `result_timeout` – timeout that was passed into `future.result(..)`

**Returns** timeout

**Return type** `float` or tuple(`connect_timeout`, `timeout`)

**result** (`timeout=None`)

Blocking call to wait for API response

**Parameters** `timeout` (`float`) – timeout in seconds to wait for response. Defaults to None to wait indefinitely.

**Returns** raw response from the server

**Return type** `dict`

`timeout_errors = [<class 'requests.exceptions.ReadTimeout'>]`

**class** `bravado.requests_client.RequestsResponseAdapter` (`requests_lib_response`)

Bases: `bravado_core.response.IncomingResponse`

Wraps a `requests.models.Response` object to provide a uniform interface to the response innards.

**headers**

**json** (`**kwargs`)

**Returns** response content in a json-like form

**Return type** `int`, `float`, `double`, `string`, `unicode`, `list`, `dict`

```
raw_bytes
reason
status_code
text
```

### 6.1.5 fido\_client Module

### 6.1.6 http\_future Module

**class** bravado.http\_future.FutureAdapter

Bases: `object`

Mimics a `concurrent.futures.Future` regardless of which client is performing the request, whether it is synchronous or actually asynchronous.

This adapter must be implemented by all bravado clients such as FidoClient or RequestsClient to wrap the object returned by their 'request' method.

**result** (*timeout=None*)

Must implement a result method which blocks on result retrieval.

**Parameters** *timeout* – maximum time to wait on result retrieval. Defaults to None which means blocking indefinitely.

**timeout\_errors** = []

**class** bravado.http\_future.HttpFuture (*future, response\_adapter, operation=None, request\_config=None*)

Bases: `object`

Wrapper for a `FutureAdapter` that returns an HTTP response.

#### Parameters

- **future** – The future object to wrap.
- **response\_adapter** (type that is a subclass of `bravado_core.response.IncomingResponse`.) – Adapter type which exposes the innards of the HTTP response in a non-http client specific way.
- **request\_config** (`RequestConfig`) – See `bravado.config.RequestConfig` and `bravado.client.REQUEST_OPTIONS_DEFAULTS`

**response** (*timeout=None, fallback\_result=None, exceptions\_to\_catch=(<class 'bravado.exception.BravadoTimeoutError'>, <class 'bravado.exception.HTTPServerError'>)*)  
Blocking call to wait for the HTTP response.

#### Parameters

- **timeout** (*float*) – Number of seconds to wait for a response. Defaults to None which means wait indefinitely.
- **fallback\_result** (*callable that takes an exception and returns a fallback swagger result*) – callable that accepts an exception as argument and returns the swagger result to use in case of errors
- **exceptions\_to\_catch** (*List/Tuple of Exception classes*.) – Exception classes to catch and call *fallback\_result* with. Has no effect if *fallback\_result* is not provided. By default, *fallback\_result* will be called for read timeout and server errors (HTTP 5XX).



**Returns** A BravadoResponse instance containing the swagger result and response metadata.

WARNING: This interface is considered UNSTABLE. Backwards-incompatible API changes may occur; use at your own risk.

**result** (*timeout=None*)

DEPRECATED: please use the *response()* method instead.

Blocking call to wait for and return the unmarshalled swagger result.

**Parameters** *timeout* (*float*) – Number of seconds to wait for a response. Defaults to None which means wait indefinitely.

**Returns** Depends on the value of *also\_return\_response* sent in to the constructor.

`bravado.http_future.raise_on_expected(http_response)`

Raise an HTTPError if the response is non-2XX and matches a response in the swagger spec.

**Parameters** *http\_response* – `bravado_core.response.IncomingResponse`

**Raises** HTTPError

`bravado.http_future.raise_on_unexpected(http_response)`

Raise an HTTPError if the response is 5XX.

**Parameters** *http\_response* – `bravado_core.response.IncomingResponse`

**Raises** HTTPError

`bravado.http_future.reraise_errors(func)`

`bravado.http_future.unmarshal_response(incoming_response, operation, response_callbacks=None)`

So the *http\_client* is finished with its part of processing the response. This hands the response over to *bravado\_core* for validation and unmarshalling and then runs any response callbacks. On success, the *swagger\_result* is available as *incoming\_response.swagger\_result*. :type *incoming\_response*: `bravado_core.response.IncomingResponse` :type *operation*: `bravado_core.operation.Operation` :type *response\_callbacks*: list of callable. See

`bravado_core.client.REQUEST_OPTIONS_DEFAULTS`.

**Raises** HTTPError - On 5XX status code, the HTTPError has minimal information. - On non-2XX status code with no matching response, the HTTPError contains a detailed error message.

- **On non-2XX status code with a matching response, the HTTPError** contains the return value.

`bravado.http_future.unmarshal_response_inner(response, op)`

Unmarshal incoming http response into a value based on the response specification. :type *response*: `bravado_core.response.IncomingResponse` :type *op*: `bravado_core.operation.Operation` :returns: value where type(value) matches *response\_spec*['schema']['type']

if it exists, None otherwise.

## 6.1.7 response Module

**class** `bravado.response.BravadoResponse(result, metadata)`

Bases: `object`

Bravado response object containing the swagger result as well as response metadata.

WARNING: This interface is considered UNSTABLE. Backwards-incompatible API changes may occur; use at your own risk.

#### Variables

- **result** – Swagger result from the server
- **metadata** (`BravadoResponseMetadata`) – metadata for this response including HTTP response

#### incoming\_response

```
class bravado.response.BravadoResponseMetadata(incoming_response, swagger_result,
                                                start_time, request_end_time, handled_exception_info, request_config)
```

Bases: `object`

HTTP response metadata.

NOTE: The *elapsed\_time* attribute might be slightly lower than the actual time spent since calling the operation object, as we only start measuring once the call to *HTTPClient.request* returns. Nevertheless, it should be accurate enough for logging and debugging, i.e. determining what went on and how much time was spent waiting for the response.

WARNING: This interface is considered UNSTABLE. Backwards-incompatible API changes may occur; use at your own risk.

#### Variables

- **start\_time** (*float*) – monotonic timestamp at which the future was created
- **request\_end\_time** (*float*) – monotonic timestamp at which we received the HTTP response
- **processing\_end\_time** (*float*) – monotonic timestamp at which processing the response ended
- **handled\_exception\_info** (*tuple*) – 3-tuple of exception class, exception instance and string representation of the traceback in case an exception was caught during request processing.

**elapsed\_time**

**headers**

**incoming\_response**

**is\_fallback\_result**

**request\_elapsed\_time**

**status\_code**

### 6.1.8 exception Module

```
exception bravado.exception.BravadoTimeoutError
```

Bases: `exceptions.Exception`

```
exception bravado.exception.ForcedFallbackResultError
```

Bases: `exceptions.Exception`

This exception will be handled if the option to force returning a fallback result is used.

```

exception bravado.exception.HTTPBadGateway (response, message=None, swagger_result=None)
    Bases: bravado.exception.HTTPServerError
    HTTP/502 - Bad Gateway
    status_code = 502

exception bravado.exception.HTTPBadRequest (response, message=None, swagger_result=None)
    Bases: bravado.exception.HTTPClientError
    HTTP/400 - Bad Request
    status_code = 400

exception bravado.exception.HTTPClientError (response, message=None, swagger_result=None)
    Bases: bravado.exception.HTTPError
    4xx responses.

exception bravado.exception.HTTPConflict (response, message=None, swagger_result=None)
    Bases: bravado.exception.HTTPClientError
    HTTP/409 - Conflict
    status_code = 409

exception bravado.exception.HTTPError (response, message=None, swagger_result=None)
    Bases: exceptions.IOError
    Unified HTTPError used across all http_client implementations.

class bravado.exception.HTTPErrorType
    Bases: type
    A metaclass for registering HTTPError subclasses.

exception bravado.exception.HTTPExpectationFailed (response, message=None, swagger_result=None)
    Bases: bravado.exception.HTTPClientError
    HTTP/417 - Expectation Failed
    status_code = 417

exception bravado.exception.HTTPFailedDependency (response, message=None, swagger_result=None)
    Bases: bravado.exception.HTTPClientError
    HTTP/424 - Failed Dependency
    status_code = 424

exception bravado.exception.HTTPForbidden (response, message=None, swagger_result=None)
    Bases: bravado.exception.HTTPClientError
    HTTP/403 - Forbidden
    status_code = 403

exception bravado.exception.HTTPFound (response, message=None, swagger_result=None)
    Bases: bravado.exception.HTTPRedirection
    HTTP/302 - Found

```

```
    status_code = 302
exception bravado.exception.HTTPGatewayTimeout (response, message=None, swagger_result=None)
    Bases: bravado.exception.HTTPServerError
    HTTP/504 - Gateway Timeout
    status_code = 504
exception bravado.exception.HTTPGone (response, message=None, swagger_result=None)
    Bases: bravado.exception.HTTPClientError
    HTTP/410 - Gone
    status_code = 410
exception bravado.exception.HTTPHTTPVersionNotSupported (response, message=None, swagger_result=None)
    Bases: bravado.exception.HTTPServerError
    HTTP/505 - HTTP Version Not Supported
    status_code = 505
exception bravado.exception.HTTPInsufficientStorage (response, message=None, swagger_result=None)
    Bases: bravado.exception.HTTPServerError
    HTTP/507 - Insufficient Storage
    status_code = 507
exception bravado.exception.HTTPInternalServerError (response, message=None, swagger_result=None)
    Bases: bravado.exception.HTTPServerError
    HTTP/500 - Internal Server Error
    status_code = 500
exception bravado.exception.HTTPLengthRequired (response, message=None, swagger_result=None)
    Bases: bravado.exception.HTTPClientError
    HTTP/411 - Length Required
    status_code = 411
exception bravado.exception.HTTPLocked (response, message=None, swagger_result=None)
    Bases: bravado.exception.HTTPClientError
    HTTP/423 - Locked
    status_code = 423
exception bravado.exception.HTTPLoopDetected (response, message=None, swagger_result=None)
    Bases: bravado.exception.HTTPServerError
    HTTP/508 - Loop Detected
    status_code = 508
exception bravado.exception.HTTPMethodNotAllowed (response, message=None, swagger_result=None)
    Bases: bravado.exception.HTTPClientError
```

HTTP/405 - Method Not Allowed

**status\_code = 405**

**exception** bravado.exception.HTTPMisdirectedRequest (response, message=None, swagger\_result=None)

Bases: *bravado.exception.HTTPClientError*

HTTP/421 - Misdirected Request

**status\_code = 421**

**exception** bravado.exception.HTTPMovedPermanently (response, message=None, swagger\_result=None)

Bases: *bravado.exception.HTTPRedirection*

HTTP/301 - Moved Permanently

**status\_code = 301**

**exception** bravado.exception.HTTPMultipleChoices (response, message=None, swagger\_result=None)

Bases: *bravado.exception.HTTPRedirection*

HTTP/300 - Multiple Choices

**status\_code = 300**

**exception** bravado.exception.HTTPNetworkAuthenticationRequired (response, message=None, swagger\_result=None)

Bases: *bravado.exception.HTTPServerError*

HTTP/511 - Network Authentication Required

**status\_code = 511**

**exception** bravado.exception.HTTPNotAcceptable (response, message=None, swagger\_result=None)

Bases: *bravado.exception.HTTPClientError*

HTTP/406 - Not Acceptable

**status\_code = 406**

**exception** bravado.exception.HTTPNotExtended (response, message=None, swagger\_result=None)

Bases: *bravado.exception.HTTPServerError*

HTTP/510 - Not Extended

**status\_code = 510**

**exception** bravado.exception.HTTPNotFound (response, message=None, swagger\_result=None)

Bases: *bravado.exception.HTTPClientError*

HTTP/404 - Not Found

**status\_code = 404**

**exception** bravado.exception.HTTPNotImplemented (response, message=None, swagger\_result=None)

Bases: *bravado.exception.HTTPServerError*

HTTP/501 - Not Implemented

**status\_code = 501**

**exception** bravado.exception.HTTPNotModified(response, message=None, swagger\_result=None)

Bases: *bravado.exception.HTTPRedirection*

HTTP/304 - Not Modified

**status\_code = 304**

**exception** bravado.exception.HTTPPayloadTooLarge(response, message=None, swagger\_result=None)

Bases: *bravado.exception.HTTPClientError*

HTTP/413 - Payload Too Large

**status\_code = 413**

**exception** bravado.exception.HTTPPaymentRequired(response, message=None, swagger\_result=None)

Bases: *bravado.exception.HTTPClientError*

HTTP/402 - Payment Required

**status\_code = 402**

**exception** bravado.exception.HTTPPermanentRedirect(response, message=None, swagger\_result=None)

Bases: *bravado.exception.HTTPRedirection*

HTTP/308 - Permanent Redirect

**status\_code = 308**

**exception** bravado.exception.HTTPPreconditionFailed(response, message=None, swagger\_result=None)

Bases: *bravado.exception.HTTPClientError*

HTTP/412 - Precondition Failed

**status\_code = 412**

**exception** bravado.exception.HTTPPreconditionRequired(response, message=None, swagger\_result=None)

Bases: *bravado.exception.HTTPClientError*

HTTP/428 - Precondition Required

**status\_code = 428**

**exception** bravado.exception.HTTPProxyAuthenticationRequired(response, message=None, swagger\_result=None)

Bases: *bravado.exception.HTTPClientError*

HTTP/407 - Proxy Authentication Required

**status\_code = 407**

**exception** bravado.exception.HTTPRangeNotSatisfiable(response, message=None, swagger\_result=None)

Bases: *bravado.exception.HTTPClientError*

HTTP/416 - Range Not Satisfiable

**status\_code = 416**

---

```

exception bravado.exception.HTTPRedirection(response, message=None, swagger_result=None)
    Bases: bravado.exception.HTTPError
    3xx responses.

exception bravado.exception.HTTPRequestHeaderFieldsTooLarge(response, message=None, swagger_result=None)
    Bases: bravado.exception.HTTPClientError
    HTTP/431 - Request Header Fields Too Large
    status_code = 431

exception bravado.exception.HTTPRequestTimeout(response, message=None, swagger_result=None)
    Bases: bravado.exception.HTTPClientError
    HTTP/408 - Request Timeout
    status_code = 408

exception bravado.exception.HTTPSeeOther(response, message=None, swagger_result=None)
    Bases: bravado.exception.HTTPRedirection
    HTTP/303 - See Other
    status_code = 303

exception bravado.exception.HTTPServerError(response, message=None, swagger_result=None)
    Bases: bravado.exception.HTTPError
    5xx responses.

exception bravado.exception.HTTPServiceUnavailable(response, message=None, swagger_result=None)
    Bases: bravado.exception.HTTPServerError
    HTTP/503 - Service Unavailable
    status_code = 503

exception bravado.exception.HTTPTemporaryRedirect(response, message=None, swagger_result=None)
    Bases: bravado.exception.HTTPRedirection
    HTTP/307 - Temporary Redirect
    status_code = 307

exception bravado.exception.HTTPTooManyRequests(response, message=None, swagger_result=None)
    Bases: bravado.exception.HTTPClientError
    HTTP/429 - Too Many Requests
    status_code = 429

exception bravado.exception.HTTPURITooLong(response, message=None, swagger_result=None)
    Bases: bravado.exception.HTTPClientError
    HTTP/414 - URI Too Long
    status_code = 414

```

```
exception bravado.exception.HTTPUnauthorized(response, message=None, swagger_result=None)
```

Bases: `bravado.exception.HTTPClientError`

HTTP/401 - Unauthorized

**status\_code** = 401

```
exception bravado.exception.HTTPUnavailableForLegalReasons(response, message=None, swagger_result=None)
```

Bases: `bravado.exception.HTTPClientError`

HTTP/451 - Unavailable For Legal Reasons

**status\_code** = 451

```
exception bravado.exception.HTTPUnprocessableEntity(response, message=None, swagger_result=None)
```

Bases: `bravado.exception.HTTPClientError`

HTTP/422 - Unprocessable Entity

**status\_code** = 422

```
exception bravado.exception.HTTPUnsupportedMediaType(response, message=None, swagger_result=None)
```

Bases: `bravado.exception.HTTPClientError`

HTTP/415 - Unsupported Media Type

**status\_code** = 415

```
exception bravado.exception.HTTPUpgradeRequired(response, message=None, swagger_result=None)
```

Bases: `bravado.exception.HTTPClientError`

HTTP/426 - Upgrade Required

**status\_code** = 426

```
exception bravado.exception.HTTPUseProxy(response, message=None, swagger_result=None)
```

Bases: `bravado.exception.HTTPRedirection`

HTTP/305 - Use Proxy

**status\_code** = 305

```
exception bravado.exception.HTTPVariantAlsoNegotiates(response, message=None, swagger_result=None)
```

Bases: `bravado.exception.HTTPServerError`

HTTP/506 - Variant Also Negotiates

**status\_code** = 506

```
bravado.exception.make_http_exception(response, message=None, swagger_result=None)
```

Return an HTTP exception class based on the response. If a specific class doesn't exist for a particular HTTP status code, a more general `HTTPError` class will be returned. :type response: `bravado_core.response.IncomingResponse` :param message: Optional string message :param swagger\_result: If the response for this `HTTPError` is

documented in the swagger spec, then this should be the result value of the response.

**Returns** An HTTP exception class that can be raised



### 6.1.9 testing Module

**class** bravado.testing.response\_mock.**BravadoResponseMock** (*result, metadata=None*)

Bases: `object`

Class that behaves like the `HttpFuture.response()` method as well as a `BravadoResponse`. Please check the documentation for further information.

**metadata**

**result**

**class** bravado.testing.response\_mock.**FallbackResultBravadoResponseMock** (*exception=BravadoTimeoutE*  
*meta-*  
*data=None*)

Bases: `object`

Class that behaves like the `HttpFuture.response()` method as well as a `BravadoResponse`. It will always call the `fallback_result` callback that's passed to the `response()` method. Please check the documentation for further information.

**metadata**

**result**

**class** bravado.testing.response\_mock.**IncomingResponseMock** (*status\_code, \*\*kwargs*)

Bases: `bravado_core.response.IncomingResponse`



### 7.1 10.0.1 (2018-06-20)

- Add helper classes (in *bravado.testing.response\_mock*s) for unit testing code using bravado - PR #375

### 7.2 10.0.0 (2018-06-15)

- Re-add ability to force returning fallback results - PR #372. Per-request configuration is now handled by the new *bravado.config.RequestConfig* class. This change requires an updated version of bravado-asyncio in case you're using that HTTP client.

### 7.3 9.3.2 (2018-06-15)

- Revert ability to force returning fallback results which was introduced in 9.3.1, since it contains backwards-incompatible changes that break third-party HTTP clients like bravado-asyncio.

### 7.4 9.3.1 (2018-06-14)

- Add ability to force returning fallback results - PR #372

### 7.5 9.3.0 (2018-06-05)

- Introduce the HTTPFuture.response API as well as support for returning a fallback result. - PR #365, #366, #367, #368

*NOTE:* Most of this API is not documented yet and is considered experimental; we're working on stabilizing it and providing developer documentation.

## 7.6 9.2.2 (2017-12-19)

- Fix msgpack import issue - PR #341. Thanks Jesse Myers for your contribution!

## 7.7 9.2.1 (2017-12-07)

- The timeout exception for the requests client should inherit from `requests.exceptions.ReadTimeout` instead of `requests.exceptions.Timeout` - PR #337

## 7.8 9.2.0 (2017-11-10)

- Support msgpack as wire format for response data - PR #323, 328, 330, 331
- Allow client to access resources for tags which are not valid Python identifier names, by adding the `SwaggerClient.get_resource` method. For example, `client.get_resource('My Pets').list_pets()` - PR #320. Thanks Craig Blaszczyk for your contribution!
- Unify timeout exception classes. You can now simply catch `bravado.exception.BravadoTimeoutError` (or `builtins.TimeoutError` if you're using Python 3.3+) - PR #321

## 7.9 9.1.1 (2017-10-10)

- Allow users to pass the `tcp_nodelay` request parameter to `FidoClient` requests - PR #319

## 7.10 9.1.0 (2017-08-02)

- Make sure HTTP header names and values are unicode strings when using the fido HTTP client. NOTE: this is a potentially backwards incompatible change if you're using the fido HTTP client and are working with response headers. It's also highly advised to not upgrade to bravado-core 4.8.0+ if you're using fido unless you're also upgrading to a bravado version that contains this change.

## 7.11 9.0.7 (2017-07-05)

- Require fido version 4.2.1 so we stay compatible to code catching `crochet.TimeoutError`

## 7.12 9.0.6 (2017-06-28)

- Don't mangle headers with bytestring values on Python 3

## 7.13 9.0.5 (2017-06-23)

- Make sure headers passed in for fetching specs are converted to str as well

## 7.14 9.0.4 (2017-06-22)

- Fix regression when passing swagger parameters of type header in `_request_options` introduced by PR #288

## 7.15 9.0.3 (2017-06-21)

- When using the fido HTTP client and passing a timeout to `result()`, make sure we throw a fido `HTTPTimeoutError` instead of a `crochet TimeoutError` when hitting the timeout.

## 7.16 9.0.2 (2017-06-12)

- `_requests_options` headers are casted to `string` to support newer version of `requests` library.

## 7.17 9.0.1 (2017-06-09)

- Convert http method to str while constructing the request to fix an issue with file uploads when using requests library versions before 2.8.

## 7.18 9.0.0 (2017-06-06)

- Add API key authentication via header to `RequestsClient`.
- Fido client is now an optional dependency. **NOTE:** if you intend to use bravado with the fido client you need to install bravado with fido extras (`pip install bravado[fido]`)

## 7.19 8.4.0 (2016-09-27)

- Remove support for Python 2.6, fixing a build failure.
- Switch from Python 3.4 to Python 3.5 for tests.

## 7.20 8.3.0 (2016-06-03)

- Bravado using Fido 3.2.0 python 3 ready

## 7.21 8.2.0 (2016-04-29)

- Bravado compliant to Fido 3.0.0
- Dropped use of concurrent futures in favor of `crochet.EventualResult`
- Workaround for bypassing a unicode bug in python `requests < 2.8.1`

## 7.22 8.1.2 (2016-04-18)

- Don't unnecessarily constrain the version of twisted when not using python 2.6

## 7.23 8.1.1 (2016-04-13)

- Removed logic to build multipart forms. Using python 'requests' instead to build the entire http request.

## 7.24 8.1.0 (2016-04-04)

- Support for YAML Swagger specs - PR #198
- Remove `pytest-mock` dependency from `requirements-dev.txt`. No longer used and it was breaking the build.
- Requires `bravado-core >= 4.2.2`
- Fix unit test for default values getting sent in the request

## 7.25 8.0.1 (2015-12-02)

- Require `twisted < 15.5.0` since Python 2.6 support was dropped

## 7.26 8.0.0 (2015-11-25)

- Support for recursive \$refs
- Support for remote \$refs e.g. Swagger 2.0 specs that span multiple json files
- Requires `bravado-core 4.0.0` which is not backwards compatible (See its [CHANGELOG](#))
- Transitively requires `swagger-spec-validator 2.0.2` which is not backwards compatible (See its [CHANGELOG](#))

## 7.27 7.0.0 (2015-10-23)

- Support per-request `response_callbacks` to enable `SwaggerClient` decorators to instrument an `IncomingResponse` post-receive. This is a non-backwards compatible change iff you have implemented a custom `HttpClient`. Consult the changes in signature to `HttpClient.request()` and `HttpFuture`'s constructor.
- Config option `also_return_response` is supported on a per-request basis.

## 7.28 6.1.1 (2015-10-19)

- Fix `IncomingResponse` subclasses to provide access to the http headers.
- Requires `bravado-core`  $\geq 3.1.0$

## 7.29 6.1.0 (2015-10-19)

- Clients can now access the HTTP response from a service call to access things like headers and status code. See [Advanced Usage](#)

## 7.30 6.0.0 (2015-10-12)

- User-defined formats are no longer global. The registration mechanism has changed and is now done via configuration. See [Configuration](#)

## 7.31 5.0.0 (2015-08-27)

- Update `ResourceDecorator` to return an operation as a `CallableOperation` instead of a function wrapper (for the docstring). This allows further decoration of the `ResourceDecorator`.

## 7.32 4.0.0 (2015-08-10)

- Consistent `bravado.exception.HTTPError` now thrown from both `Fido` and `Requests` http clients.
- `HTTPError` refactored to contain an optional detailed message and Swagger response result.

## 7.33 3.0.0 (2015-08-03)

- Support passing in `connect_timeout` and `timeout` via `_request_options` to the `Fido` and `Requests` clients
- Timeout in `HTTPFuture` now defaults to `None` (wait indefinitely) instead of 5s. You should make sure any calls to `http_future.result(..)` without a timeout are updated accordingly.

## 7.34 2.1.0 (2015-07-20)

- Add warning for deprecated operations

## 7.35 2.0.0 (2015-07-13)

- Assume responsibility for http invocation (used to be in `bravado-core`)

### **7.36 1.1.0 (2015-07-06)**

- Made bravado compatible with Py34

### **7.37 1.0.0 (2015-06-26)**

- Fixed petstore demo link
- Pick up bug fixes from bravado-core 1.1.0

### **7.38 1.0.0-rc2 (2015-06-01)**

- Renamed ResponseLike to IncomingResponse to match bravado-core

### **7.39 1.0.0-rc1 (2015-05-13)**

- Initial version - large refactoring/rewrite of swagger-py 0.7.5 to support Swagger 2.0



## CHAPTER 8

---

### Indices and tables

---

- `genindex`
- `modindex`



### b

`bravado.client`, [23](#)  
`bravado.config`, [25](#)  
`bravado.exception`, [30](#)  
`bravado.http_future`, [28](#)  
`bravado.requests_client`, [25](#)  
`bravado.response`, [29](#)  
`bravado.testing.response_mocks`, [37](#)



## A

additional\_properties (bravado.config.RequestConfig attribute), 25  
also\_return\_response (bravado.config.RequestConfig attribute), 25  
ApiKeyAuthenticator (class in bravado.requests\_client), 25  
apply() (bravado.requests\_client.ApiKeyAuthenticator method), 26  
apply() (bravado.requests\_client.Authenticator method), 26  
apply() (bravado.requests\_client.BasicAuthenticator method), 26  
apply\_authentication() (bravado.requests\_client.RequestsClient method), 26  
authenticated\_request() (bravado.requests\_client.RequestsClient method), 26  
Authenticator (class in bravado.requests\_client), 26

## B

BasicAuthenticator (class in bravado.requests\_client), 26  
bravado.client (module), 23  
bravado.config (module), 25  
bravado.exception (module), 30  
bravado.http\_future (module), 28  
bravado.requests\_client (module), 25  
bravado.response (module), 29  
bravado.testing.response\_mock (module), 37  
BravadoConfig (class in bravado.config), 25  
BravadoResponse (class in bravado.response), 29  
BravadoResponseMetadata (class in bravado.response), 30  
BravadoResponseMock (class in bravado.testing.response\_mock), 37  
BravadoTimeoutError, 30  
build\_timeout() (bravado.requests\_client.RequestsFutureAdapter method), 27

## C

CallableOperation (class in bravado.client), 24  
connect\_timeout (bravado.config.RequestConfig attribute), 25  
construct\_params() (in module bravado.client), 24  
construct\_request() (in module bravado.client), 25

## E

elapsed\_time (bravado.response.BravadoResponseMetadata attribute), 30

## F

FallbackResultBravadoResponseMock (class in bravado.testing.response\_mock), 37  
force\_fallback\_result (bravado.config.RequestConfig attribute), 25  
ForcedFallbackResultError, 30  
from\_config\_dict() (bravado.config.BravadoConfig static method), 25  
from\_spec() (bravado.client.SwaggerClient class method), 24  
from\_url() (bravado.client.SwaggerClient class method), 24  
FutureAdapter (class in bravado.http\_future), 28

## G

get\_model() (bravado.client.SwaggerClient method), 24

## H

headers (bravado.config.RequestConfig attribute), 25  
headers (bravado.requests\_client.RequestsResponseAdapter attribute), 27  
headers (bravado.response.BravadoResponseMetadata attribute), 30  
HTTPBadGateway, 30  
HTTPBadRequest, 31  
HTTPClientError, 31  
HTTPConflict, 31  
HTTPError, 31

[HTTPErrorType](#) (class in [bravado.exception](#)), 31  
[HTTPExpectationFailed](#), 31  
[HTTPFailedDependency](#), 31  
[HTTPForbidden](#), 31  
[HTTPFound](#), 31  
[HttpFuture](#) (class in [bravado.http\\_future](#)), 28  
[HTTPGatewayTimeout](#), 32  
[HTTPGone](#), 32  
[HTTPHTTPVersionNotSupported](#), 32  
[HTTPInsufficientStorage](#), 32  
[HTTPInternalServerError](#), 32  
[HTTPLengthRequired](#), 32  
[HTTPLocked](#), 32  
[HTTPLoopDetected](#), 32  
[HTTPMethodNotAllowed](#), 32  
[HTTPMisdirectedRequest](#), 33  
[HTTPMovedPermanently](#), 33  
[HTTPMultipleChoices](#), 33  
[HTTPNetworkAuthenticationRequired](#), 33  
[HTTPNotAcceptable](#), 33  
[HTTPNotExtended](#), 33  
[HTTPNotFound](#), 33  
[HTTPNotImplemented](#), 33  
[HTTPNotModified](#), 34  
[HTTPPayloadTooLarge](#), 34  
[HTTPPaymentRequired](#), 34  
[HTTPPermanentRedirect](#), 34  
[HTTPPreconditionFailed](#), 34  
[HTTPPreconditionRequired](#), 34  
[HTTPProxyAuthenticationRequired](#), 34  
[HTTPRangeNotSatisfiable](#), 34  
[HTTPRedirection](#), 34  
[HTTPRequestHeaderFieldsTooLarge](#), 35  
[HTTPRequestTimeout](#), 35  
[HTTPSeeOther](#), 35  
[HTTPServerError](#), 35  
[HTTPServiceUnavailable](#), 35  
[HTTPTemporaryRedirect](#), 35  
[HTTPTooManyRequests](#), 35  
[HTTPUnauthorized](#), 35  
[HTTPUnavailableForLegalReasons](#), 36  
[HTTPUnprocessableEntity](#), 36  
[HTTPUnsupportedMediaType](#), 36  
[HTTPUpgradeRequired](#), 36  
[HTTPURITooLong](#), 35  
[HTTPUseProxy](#), 36  
[HTTPVariantAlsoNegotiates](#), 36

## I

[incoming\\_response](#) ([bravado.response.BravadoResponse](#) attribute), 30  
[incoming\\_response](#) ([bravado.response.BravadoResponseMetadata](#) attribute), 30

[IncomingResponseMock](#) (class in [bravado.testing.response\\_mock](#)), 37  
[inject\\_headers\\_for\\_remote\\_refs\(\)](#) (in module [bravado.client](#)), 25  
[is\\_fallback\\_result](#) ([bravado.response.BravadoResponseMetadata](#) attribute), 30

## J

[json\(\)](#) ([bravado.requests\\_client.RequestsResponseAdapter](#) method), 27

## M

[make\\_http\\_exception\(\)](#) (in module [bravado.exception](#)), 36  
[matches\(\)](#) ([bravado.requests\\_client.Authenticator](#) method), 26  
[metadata](#) ([bravado.testing.response\\_mock.BravadoResponseMock](#) attribute), 37  
[metadata](#) ([bravado.testing.response\\_mock.FallbackResultBravadoResponseMock](#) attribute), 37

## R

[raise\\_on\\_expected\(\)](#) (in module [bravado.http\\_future](#)), 29  
[raise\\_on\\_unexpected\(\)](#) (in module [bravado.http\\_future](#)), 29  
[raw\\_bytes](#) ([bravado.requests\\_client.RequestsResponseAdapter](#) attribute), 27  
[reason](#) ([bravado.requests\\_client.RequestsResponseAdapter](#) attribute), 28  
[request\(\)](#) ([bravado.requests\\_client.RequestsClient](#) method), 26  
[request\\_elapsed\\_time](#) ([bravado.response.BravadoResponseMetadata](#) attribute), 30  
[RequestConfig](#) (class in [bravado.config](#)), 25  
[RequestsClient](#) (class in [bravado.requests\\_client](#)), 26  
[RequestsFutureAdapter](#) (class in [bravado.requests\\_client](#)), 27  
[RequestsResponseAdapter](#) (class in [bravado.requests\\_client](#)), 27  
[reraise\\_errors\(\)](#) (in module [bravado.http\\_future](#)), 29  
[ResourceDecorator](#) (class in [bravado.client](#)), 24  
[response\(\)](#) ([bravado.http\\_future.HttpFuture](#) method), 28  
[response\\_callbacks](#) ([bravado.config.RequestConfig](#) attribute), 25  
[result](#) ([bravado.testing.response\\_mock.BravadoResponseMock](#) attribute), 37  
[result](#) ([bravado.testing.response\\_mock.FallbackResultBravadoResponseMock](#) attribute), 37  
[result\(\)](#) ([bravado.http\\_future.FutureAdapter](#) method), 28  
[result\(\)](#) ([bravado.http\\_future.HttpFuture](#) method), 29  
[result\(\)](#) ([bravado.requests\\_client.RequestsFutureAdapter](#) method), 27

## S

- `separate_params()` (bravado.requests\_client.RequestsClient static method), 27
- `set_api_key()` (bravado.requests\_client.RequestsClient method), 27
- `set_basic_auth()` (bravado.requests\_client.RequestsClient method), 27
- `status_code` (bravado.exception.HTTPBadGateway attribute), 31
- `status_code` (bravado.exception.HTTPBadRequest attribute), 31
- `status_code` (bravado.exception.HTTPConflict attribute), 31
- `status_code` (bravado.exception.HTTPExpectationFailed attribute), 31
- `status_code` (bravado.exception.HTTPFailedDependency attribute), 31
- `status_code` (bravado.exception.HTTPForbidden attribute), 31
- `status_code` (bravado.exception.HTTPFound attribute), 31
- `status_code` (bravado.exception.HTTPGatewayTimeout attribute), 32
- `status_code` (bravado.exception.HTTPGone attribute), 32
- `status_code` (bravado.exception.HTTPHTTPVersionNotSupported attribute), 32
- `status_code` (bravado.exception.HTTPInsufficientStorage attribute), 32
- `status_code` (bravado.exception.HTTPInternalServerError attribute), 32
- `status_code` (bravado.exception.HTTPLengthRequired attribute), 32
- `status_code` (bravado.exception.HTTPLocked attribute), 32
- `status_code` (bravado.exception.HTTPLoopDetected attribute), 32
- `status_code` (bravado.exception.HTTPMethodNotAllowed attribute), 33
- `status_code` (bravado.exception.HTTPMisdirectedRequest attribute), 33
- `status_code` (bravado.exception.HTTPMovedPermanently attribute), 33
- `status_code` (bravado.exception.HTTPMultipleChoices attribute), 33
- `status_code` (bravado.exception.HTTPNetworkAuthenticationRequired attribute), 33
- `status_code` (bravado.exception.HTTPNotAcceptable attribute), 33
- `status_code` (bravado.exception.HTTPNotExtended attribute), 33
- `status_code` (bravado.exception.HTTPNotFound attribute), 33
- `status_code` (bravado.exception.HTTPNotImplemented attribute), 33
- `status_code` (bravado.exception.HTTPNotModified attribute), 34
- `status_code` (bravado.exception.HTTPPayloadTooLarge attribute), 34
- `status_code` (bravado.exception.HTTPPaymentRequired attribute), 34
- `status_code` (bravado.exception.HTTPPermanentRedirect attribute), 34
- `status_code` (bravado.exception.HTTPPreconditionFailed attribute), 34
- `status_code` (bravado.exception.HTTPPreconditionRequired attribute), 34
- `status_code` (bravado.exception.HTTPProxyAuthenticationRequired attribute), 34
- `status_code` (bravado.exception.HTTPRangeNotSatisfiable attribute), 34
- `status_code` (bravado.exception.HTTPRequestHeaderFieldsTooLarge attribute), 35
- `status_code` (bravado.exception.HTTPRequestTimeout attribute), 35
- `status_code` (bravado.exception.HTTPSeeOther attribute), 35
- `status_code` (bravado.exception.HTTPServiceUnavailable attribute), 35
- `status_code` (bravado.exception.HTTPTemporaryRedirect attribute), 35
- `status_code` (bravado.exception.HTTPTooManyRequests attribute), 35
- `status_code` (bravado.exception.HTTPUnauthorized attribute), 36
- `status_code` (bravado.exception.HTTPUnavailableForLegalReasons attribute), 36
- `status_code` (bravado.exception.HTTPUnprocessableEntity attribute), 36
- `status_code` (bravado.exception.HTTPUnsupportedMediaType attribute), 36
- `status_code` (bravado.exception.HTTPUpgradeRequired attribute), 36
- `status_code` (bravado.exception.HTTPURITooLong attribute), 35
- `status_code` (bravado.exception.HTTPUseProxy attribute), 36
- `status_code` (bravado.exception.HTTPVariantAlsoNegotiates attribute), 36
- `status_code` (bravado.requests\_client.RequestsResponseAdapter attribute), 28
- `status_code` (bravado.response.BravadoResponseMetadata attribute), 30
- `SwaggerClient` (class in bravado.client), 24

## T

- `text` (bravado.requests\_client.RequestsResponseAdapter attribute), 28
- `timeout` (bravado.config.RequestConfig attribute), 25

`timeout_errors` (`bravado.http_future.FutureAdapter` attribute), [28](#)  
`timeout_errors` (`bravado.requests_client.RequestsFutureAdapter` attribute), [27](#)

## U

`unmarshal_response()` (in module `bravado.http_future`), [29](#)  
`unmarshal_response_inner()` (in module `bravado.http_future`), [29](#)  
`use_msgpack` (`bravado.config.RequestConfig` attribute), [25](#)