koi Documentation

Release 0.4.5

Open Permissions Platform Coalition

March 28, 2017

Contents

Koi is the tornado library of the open permissions platform. This is the documentation of the Python API intended for developers. To have more information on the deployment of the library and usage please refer to our repository http://github.com/openpermissions/koi .

Koi is the tornado library of the open permissions platform. This is the documentation of the Python API intended for developers. To have more information on the deployment of the library and usage please refer to our repository http://github.com/openpermissions/koi .

Koi is the tornado library of the open permissions platform. This is the documentation of the Python API intended for developers. To have more information on the deployment of the library and usage please refer to our repository http://github.com/openpermissions/koi .

Koi is the tornado library of the open permissions platform. This is the documentation of the Python API intended for developers. To have more information on the deployment of the library and usage please refer to our repository http://github.com/openpermissions/koi .

Contents:

CHAPTER 1

koi package

Submodules

koi.auth module

Provides a auth_required decorator to check a request is authenticated, and an authorised decorator to check a request is authorised

koi.auth.**auth_optional**(*validator*)

Decorate a RequestHandler or method to accept optional authentication token

If decorating a coroutine make sure coroutine decorator is first. eg.:

```
class Handler(tornado.web.RequestHandler):

    @auth_required(validator)
    @coroutine
    def get(self):
        pass
```

Parameters **validator** – a coroutine that will validate the token and return True/False

koi.auth.**auth_required**(*validator*)

Decorate a RequestHandler or method to require that a request is authenticated

If decorating a coroutine make sure coroutine decorator is first. eg.:

```
class Handler(tornado.web.RequestHandler):

    @auth_required(validator)
    @coroutine
    def get(self):
        pass
```

Parameters **validator** – a coroutine that will validate the token and return True/False

koi.auth.**authorized**(*validator*)

Decorate a RequestHandler or method to require that a request is authorized

If decorating a coroutine make sure coroutine decorator is first. eg.:

```
class Handler(tornado.web.RequestHandler):

    @authorized(validator)
    @coroutine
    def get(self):
        pass
```

Parameters **validator** – a coroutine that will authorize the user associated with the token and return True/False

koi.base module

class koi.base.**AuthHandler**(*application*, *request*, ***kwargs*)
Bases: tornado.web.RequestHandler

METHOD_ACCESS = {'HEAD': 'r', 'GET': 'r', 'PATCH': 'w', 'PUT': 'w', 'POST': 'w', 'DELETE': 'w'}

READ_ACCESS = 'r'

READ_WRITE_ACCESS = 'rw'

UNAUTHENTICATED_ACCESS = 'unauthenticated'

WRITE_ACCESS = 'w'

endpoint_access(*method*)
Determine access level needed for endpoint :param method: The request verb :return: String representing access type.

prepare(**args*, ***kwargs*)
If OAuth verification is required, validate provided token

Raise HTTPError if token does not have access

verify_token(**args*, ***kwargs*)
Check the token bearer is permitted to access the resource

Parameters

- **token** – Access token
- **requested_access** – the access level the client has requested

Returns boolean

class koi.base.**BaseHandler**(*application*, *request*, ***kwargs*)
Bases: koi.base.AuthHandler, koi.base.CorsHandler, koi.base.JsonHandler

class koi.base.**CorsHandler**(*application*, *request*, ***kwargs*)
Bases: tornado.web.RequestHandler

Shared code for handling CORS

options(**args*, ***kwargs*)
Default OPTIONS response

If the 'cors' option is True, will respond with an empty response and set the 'Access-Control-Allow-Headers' and 'Access-Control-Allow-Methods' headers

set_default_headers()
Set the default headers

If the 'cors' option is True, will set the 'Access-Control-Allow-Origin' header to '*'

class koi.base.**JsonHandler**(*application*, *request*, ***kwargs*)
Bases: tornado.web.RequestHandler

Shared code for handling JSON requests

get_json_body(*required=None*, *validators=None*)
Get JSON from the request body

Parameters **required** – optionally provide a list of keys that should be

in the JSON body (raises a 400 HTTPError if any are missing) :param validator: optionally provide a dictionary of items that should be in the body with a method that validates the item. The method must be synchronous and return a boolean, no exceptions. :raises: HTTPError

write_error(*status_code*, ***kwargs*)
Override *write_error* in order to output JSON errors

Parameters **status_code** – the response's status code, e.g. 500

koi.commands module

class koi.commands.**Command**(*main*, *conf_dir=None*, *commands_dir=None*, ***kwargs*)
Bases: click.core.MultiCommand

A MultiCommand that extends the defaults click.Group with commands in a service.

Parameters

- **main** – the main function to run the service. This function will be called if the CLI is not provided with any subcommands (e.g. *python accounts*).
- **conf_dir** – path to the service's tornado configuration directory
- **commands_dir** – path to the commands directory. Python modules within the module will be attached to this CLI as a subcommand. Each module needs to have a *cli* variable in it's namespace which is a click.Command, click.Group or click.MultiCommand instance.

get_command(*ctx*, *name*)
Get the command from either the commands dir or default group

list_commands(*ctx*)
List commands from the commands dir and default group

koi.commands.**cli**(*main*, *conf_dir=None*, *commands_dir=None*)
Convenience function for initialising a Command CLI

For parameter definitions see Command

koi.commands.**run**(*func*)
Execute the provided function if there are no subcommands

koi.configure module

Configure ssl server and client

class koi.configure.**ErrorHandler**(*application*, *request*, ***kwargs*)
Bases: tornado.web.RequestHandler

prepare()

class koi.configure.**RequestFilter**(*request*)
Bases: logging.Filter

filter(*record*)

koi.configure.**configure_syslog**(*request=None*, *logger=None*, *exceptions=False*)
Configure syslog logging channel. It is turned on by setting *syslog_host* in the config file. The port default to 514 can be overridden by setting *syslog_port*.

Parameters

- **request** – tornado.httputil.HTTPServerRequest instance
- **exceptions** – boolean - This indicates if we should raise exceptions encountered in the logging system.

koi.configure.**define_options**(*default_conf*)
Define the options from default.conf dynamically

koi.configure.**load_config**(*conf_dir*)
Use default.conf as the definition of options with default values using tornado.options.define. Then overrides the values from: local.conf. This mapping allows to access the application configuration across the application.

Parameters **conf_dir** – path to configuration directory

koi.configure.**load_config_file**(*conf_dir*)

koi.configure.**log_config**()
Logs the config used to start the application

koi.configure.**log_formatter**(*request=None*)
Log formatter used in our syslog

Parameters **request** – a request object

Returns logging.Formatter

koi.configure.**make_application**(*version*, *app_name*, *app_urls*, *kwargs=None*)
Loads the routes and starts the server

Parameters

- **version** – the application version
- **app_name** – the application name
- **app_urls** – a list of application endpoints
- **kwargs** – dictionary of options

Returns tornado.web.Application instance

koi.configure.**make_server**(*application*, *conf_dir=None*)
Configure the server return the server instance

koi.configure.**ssl_server_options**()
ssl options for tornado https server these options are defined in each application's default.conf file if left empty, use the self generated keys and certificates included in this package. this function is backward compatible with python version lower than 2.7.9 where ssl.SSLContext is not available.

koi.constants module

koi.exceptions module

exception koi.exceptions.**HTTPError**(*status_code*, *errors*, ***kwargs*)
Bases: tornado.web.HTTPError

Subclass of tornado.web.HTTPError.

Raise a HTTPError to respond to a request with an error. The base.BaseHandler will use the information in the exception to form a JSON response in our standard error format.

Parameters

- **status_code** – The status code that should be used in the HTTP response
- **errors** – Error messages
- **kwargs** – optionally include a source of the error (defaults to the service's name)

koi.keygen module

koi.keygen.**argument_parser**()
process the command line arguments. :returns: an ArgumentParser object.

koi.keygen.**call_openssl**(*cmd*, *message*, *silent=False*)
call openssl :param cmd: a string of command send to openssl :param message: a string to print out if not silent :param silent: a boolean for whether to suppress output from openssl

koi.keygen.**check_cert**(*certfile*)
output the text format of the certificate :param filepath: file path to the ssl certificate :returns: string

koi.keygen.**check_key_cert_match**(*keyfile*, *certfile*)
check if the ssl key matches the certificate :param keyfile: file path to the ssl key :param certfile: file path to the ssl certificate :returns: true or false

koi.keygen.**gen_ca_cert**(*filename*, *dirname*, *days*, *silent=False*)
generate a CA key and certificate key pair. :param filename: prefix for the key and cert file :param dirname: name of the directory :param days: days of the certificate being valid :param silent: whether to suppress output

koi.keygen.**gen_cert_request**(*filepath*, *keyfile*, *config*, *silent=False*)
generate certificate request :param filepath: file path to the certificate request :param keyfile: file path to the private key :param silent: whether to suppress output

koi.keygen.**gen_non_ca_cert**(*filename*, *dirname*, *days*, *ip_list*, *dns_list*, *ca_crt*, *ca_key*, *silent=False*)
generate a non CA key and certificate key pair signed by the private CA key and crt. :param filename: prefix for the key and cert file :param dirname: name of the directory :param days: days of the certificate being valid :ip_list: a list of ip address to be included in the certificate :dns_list: a list of dns names to be included in the certificate :ca_key: file path to the CA key :ca_crt: file path to the CA crt :param silent: whether to suppress output

koi.keygen.**gen_private_key**(*filepath*, *silent=False*)
generate ssl private key :param filepath: file path to the key file :param silent: whether to suppress output

koi.keygen.**gen_self_signed_cert**(*filepath*, *keyfile*, *days*, *silent=False*)
generate self signed ssl certificate, i.e. a private CA certificate :param filepath: file path to the key file :param keyfile: file path to the private key :param days: valid duration for the certificate :param silent: whether to suppress output

koi.keygen.**main**(*argv=None*)

koi.keygen.**sign_cert_request**(*filepath*, *cert_req*, *ca_crt*, *ca_key*, *days*, *extfile*, *silent=False*)
generate self signed ssl certificate, i.e. a private CA certificate :param filepath: file path to the key file :param keyfile: file path to the private key :param days: valid duration for the certificate :param silent: whether to suppress output

koi.test_helpers module

koi.test_helpers.**gen_test**(*func*)
Helper for running async tests, based on the tornado.testing.gen_test decorator. It wraps the test function with tornado.gen.coroutine and initialises an IOLoop to run the async code using IOLoop.run_sync NOTE: if using this with the mock.patch decorator apply gen_test first, otherwise the patches won't work. ANOTHER NOTE: if you don't yield when calling coroutines in your test you can get false positives, just like any other place where you don't call coroutines correctly. It's always a good idea to see your test fail so you know it's testing something.

koi.test_helpers.**make_future**(*result*)
Create a *tornado.concurrent.Future* that returns *result*

Useful for adding a return value to a mocked coroutine, for example:

```
mock = Mock()
mock.func.return_value = test_helpers.make_future('test')
result = IOLoop.instance().run_sync(mock.func)

assert result == 'test'
```

Parameters **result** – the Future's result

Returns tornado.concurrent.Future

koi.utils module

Useful utils.

koi.utils.**add_path_part**(**args*, ***kwargs*)
replace the variables in a url template with regex named groups :param url: string of a url template :param regex: regex of the named group :returns: regex

koi.utils.**add_prefix**(*endpoints*, *prefix*, *kwargs=None*)

koi.utils.**listify**(**args*)
Convert args to a list, unless there's one arg and it's a function, then acts a decorator.

koi.utils.**make_endpoints**(**args*, ***kwargs*)
Returns a redirect handler and all endpoints with a version prefix added.

Parameters

- **version** – the application version
- **name** – the application name
- **endpoints** – a list of application endpoints
- **kwargs** – an optional dictionary to populate placeholders in endpoints

:returns:list of endpoints

koi.utils.**sanitise_capabilities**(*capabilities*)
Makes sure dictionary of capabilities includes required options, and does not include protected ones. :param capabilities: :return: dict

koi.utils.**stringify**(**args*)
Joins args to build a string, unless there's one arg and it's a function, then acts a decorator.

koi.utils.**tuplify**(**args*)
Convert args to a tuple, unless there's one arg and it's a function, then acts a decorator.

Module contents

CHAPTER 2

Indices and tables

- *genindex*
- *modindex*
- *search*

k

R

S

T

U

V

W