Prompy Documentation

T4rk

Jun 30, 2018

Contents

CHAPTER 1

prompy

1.1 prompy package

1.1.1 Subpackages

1.1.1.1 prompy.networkio package

prompy.networkio.call_factory module

call_factory

Meta web api wrapper.

Usage:

```
from prompy.networkio.call_factory import CallRoute, Caller
from prompy.threadio.promise_queue import PromiseQueuePool


class Api(Caller):
    def call_home(self, **kwargs):
        return CallRoute('/')

    def call_data(self, **kwargs):
        return CallRoute('/data', method='POST')


pool = PromiseQueuePool(start=True)
api = Api(base_url='http://localhost:5000', promise_container=pool)
api.call_data(data={'num': 6}).then(print).catch(print)
```

class prompy.networkio.call_factory.**CallRoute**(*route*, *method='GET'*, *content_type='application/json'*)

Bases: object

Route object used by *Caller*

__init__(*route*, *method='GET'*, *content_type='application/json'*)
Route data object with format methods.

Parameters

- **route** (str) – url to call, with optional templating.
- **method** (str) – http method
- **content_type** (str) –

format_data(*data*, *encoding='utf-8'*)
Serialize the data according to content-type.

Parameters

- **encoding** –
- **data** (Union[Dict[~KT, ~VT], List[~T], str, None]) –

Returns

format_route_params(**args*)
Format the route params.

Example

```
route = CallRoute('/user/<user>')
r = route.format_route_params(user='bob')
```

Returns

class prompy.networkio.call_factory.**Caller**(*base_url=''*, *promise_container=None*, *prom_type=prompy.promise.Promise*, *prom_args=None*)

Bases: object

Wraps all method starting with *call_* with a call.

route methods must:

- return a CallRoute object.
- kwargs must be there if you want Caller.call and route params kwargs.

__init__(*base_url=''*, *promise_container=None*, *prom_type=prompy.promise.Promise*, *prom_args=None*)

Parameters

- **base_url** (str) –
- **promise_container** (Optional[*BasePromiseContainer*]) –
- **prom_type** –
- **prom_args** (Optional[dict]) –

after_call(*route*, *route_params*, *params*, *result*, *error*)
global after call callback

Parameters

- **route** (*CallRoute*) – The route that was called.

- **route_params** (list) – The params of the route if any
- **params** (dict) – The url params
- **result** (Any) – The result of the call if any
- **error** (Any) – The error of the call if any

Returns

before_call(*route*, *route_params*, *params*)
global before call callback.

Parameters

- **route** (*CallRoute*) – The route that was called.
- **route_params** (list) – The params of the route if any
- **params** (dict) – The url params

Returns

call(*route*, *route_params=None*, *params=None*, *headers=None*, *origin_req_host=None*, *unverifiable=False*, *data=None*, ***kwargs*)
Call a route, used by the wrapped route methods.

Parameters

- **route** (*CallRoute*) –
- **route_params** (Optional[list]) –
- **params** (Optional[dict]) –
- **headers** (Optional[dict]) –
- **origin_req_host** –
- **unverifiable** (bool) –
- **data** –
- **kwargs** –

Return type *Promise*[~PromiseReturnType]

Returns

prompy.networkio.urlcall module

prompy.networkio.urlcall.**get**(*url*, *params=None*, *prom_type=prompy.promise.Promise*, ***kwargs*)

Return type *Promise*[]

prompy.networkio.urlcall.**json_call**(*url*, *payload=None*, *encoding='UTF-8'*, *prom_type=prompy.promise.Promise*, *headers=None*, ***kwargs*)
Auto encode payload and decode response in json.

Return type *Promise*[]

prompy.networkio.urlcall.**post**(*url*, *data=None*, *prom_type=prompy.promise.Promise*, ***kwargs*)

Return type *Promise*[]

prompy.networkio.urlcall.**put**(*url*, *data*, *prom_type=prompy.promise.Promise*, ***kwargs*)

Return type *Promise*[]

prompy.networkio.urlcall.**url_call**(*url*, *data=None*, *headers=None*, *origin_req_host=None*, *unverifiable=False*, *method=None*, *content_mapper=<function default_content_mapper>*, *prom_type=prompy.promise.Promise*, ***kwargs*)

Base http call using urllib.

Parameters

- **url** –
- **data** –
- **headers** –
- **origin_req_host** –
- **unverifiable** –
- **method** –
- **content_mapper** (Callable[[str, str, str], Any]) –
- **prom_type** –
- **kwargs** –

Return type *Promise*[]

Returns A promise to resolve with a response.

1.1.1.2 prompy.processio package

prompy.processio.process_promise module

Experimental multiprocess promise.

class prompy.processio.process_promise.**ProcessPromise**(*starter*, *namespace=None*, **args*, ***kwargs*)

Bases: *prompy.promise.Promise*

Experimental Promise for a multiprocessing backend. Should only use for long running functions.

Closures are not serialized properly, only their values are kept.

This goes for starter and callbacks: * Objects need to be marshal compatible. * Need to import any module at function level.

__init__(*starter*, *namespace=None*, **args*, ***kwargs*)

Promise takes at least a starter method with params to this promise resolve and reject. Does not call exec by default but with start_now the execution will be synchronous.

Parameters

- **starter** (Callable[[Callable, Callable], None]) – otherwise known as executor.
- **then** – initial resolve callback
- **catch** – initial catch callback
- **complete** – initial complete callback

- **raise_again** – raise the rejection error again.
- **start_now** –
- **results_buffer_size** – number of results to keep in the buffer.

catch(*func*)
Add a callback to rejection

Parameters **func** (Callable[[Exception], None]) –

Returns

exec()
Execute the starter method.

Returns

reject(*error*)
Reject the promise.

Parameters **error** (Exception) –

Returns

resolve(*result*)
Resolve the promise, called by executor.

Parameters **result** (*~PromiseReturnType*) –

Returns

then(*func*)
Add a callback to resolve

Parameters **func** (Callable[[~PromiseReturnType], None]) – callback to resolve

Returns

prompy.processio.process_containers module

Experimental multiprocessing promise containers.

class prompy.processio.process_containers.**ProcessPromiseQueue**(*on_idle=None*, *max_idle=2*, *poll_time=0.01*, *error_list=None*, *idle_check=False*, *raise_again=True*)
Bases: *prompy.container.BasePromiseContainer*

A queue for a process promise.

Usage: *multiprocess.Process(target=ProcessPromiseQueue.run)*

__init__(*on_idle=None*, *max_idle=2*, *poll_time=0.01*, *error_list=None*, *idle_check=False*, *raise_again=True*)
Queue initializer.

Parameters

- **on_idle** (Optional[Callable]) – callback to call when the queue is idle
- **max_idle** (float) – max time the queue can be idling.

- **poll_time** (float) – the frequency of queue timeouts.
- **error_list** (Optional[<bound method BaseContext.Queue of <multiprocessing.context.DefaultContext object at 0x7f6e8c472748>>]) – a multiprocess container to exchange errors.
- **idle_check** (bool) – to use the idle timeout or not.
- **raise_again** (bool) – to raise errors again after catch (stop the queue).

add_promise(*promise*)
Add a promise to the container.

Parameters **promise** (*ProcessPromise*[]) –

Returns

errors

id

Return type int

num_tasks
The number of promise still to resolve.

Return type int

run()

running

class prompy.processio.process_containers.**PromiseProcessPool**(*pool_size=10*, *queue_options=None*)

Bases: *prompy.container.BasePromiseRunner*

A pool of PromiseQueue to add promise to.

__init__(*pool_size=10*, *queue_options=None*)

Parameters

- **pool_size** – number of processes that will be spawned.
- **queue_options** – options to give to spawned queue

add_promise(*promise*)
Add a promise to the container.

Parameters **promise** (*ProcessPromise*[]) –

Returns

get_errors()
Get all the errors from processes, they are consumed.

num_tasks
Sum of all tasks still in queue.

start()

stop()

1.1.1.3 prompy.promio package

prompy.promio.csvio module

prompy.promio.csvio.**read_csv**(*file*, *newline=''*, *reader_args=None*, *prom_type=prompy.promise.Promise*, ***kwargs*)

Return type *Promise*[~PromiseReturnType]

prompy.promio.csvio.**write_csv**(*file*, *data*, *prom_type=prompy.promise.Promise*, ***kwargs*)

Return type *Promise*[~PromiseReturnType]

prompy.promio.fileio module

Promise creators to deal with files.

Read, write, delete, compress, decompress, walk.

Example

```
from prompy.threadio.tpromise import TPromise
from prompy.promio import fileio

filename = 'myfile'

f = fileio.write_file(filename, 'content', prom_type=TPromise)
f.then(lambda _: fileio.read_file(filename).then(lambda data: print(data)))
```

prompy.promio.fileio.**compress_directory**(*directory*, *destination*, *archive_format='zip'*, *root_dir='.'*, *prom_type=prompy.promise.Promise*, ***kwargs*)

Parameters

- **directory** (str) –
- **destination** (str) –
- **archive_format** (str) –
- **root_dir** (str) –
- **prom_type** –
- **kwargs** –

Return type *Promise*[~PromiseReturnType]

Returns

prompy.promio.fileio.**decompress**(*filename*, *destination*, *archive_format='zip'*, *prom_type=prompy.promise.Promise*, ***kwargs*)

Parameters

- **filename** (str) –
- **destination** (str) –
- **archive_format** (str) –
- **prom_type** –

- **kwargs** –

Return type *Promise*[~PromiseReturnType]

Returns

prompy.promio.fileio.**delete_file**(*file*, *prom_type=prompy.promise.Promise*, ***kwargs*)

Return type *Promise*[~PromiseReturnType]

prompy.promio.fileio.**read_file**(*file*, *mode='r'*, *prom_type=prompy.promise.Promise*, ***kwargs*)

Read a file in a promise.

Parameters

- **file** (str) – to open
- **mode** – open mode ('r', 'rb')
- **prom_type** – Type of the promise to instantiate.
- **kwargs** – kwargs of the promise initializer.

Return type *Promise*[~PromiseReturnType]

Returns Promise that will resolve with the content of the file.

prompy.promio.fileio.**walk**(*directory*, *filter_directories=None*, *filter_filename=None*, *on_found=None*, *prom_type=prompy.promise.Promise*, ***kwargs*)

Resolve a list of paths that were walked.

Parameters

- **directory** (str) – path to walk.
- **on_found** – called for each path that was found.
- **filter_directories** (Optional[str]) – a regex filter to exclude directories.
- **filter_filename** (Optional[str]) – a regex filter to exclude filenames.
- **prom_type** – Type of the promise to instantiate.
- **kwargs** – kwargs of the promise initializer.

Return type *Promise*[]

Returns

prompy.promio.fileio.**write_file**(*file*, *content*, *mode='w'*, *prom_type=prompy.promise.Promise*, ***kwargs*)

Write to a file and resolve when it's done.

Parameters

- **file** (str) – to open.
- **content** (Any) – to write.
- **mode** (str) – open mode ('w', 'wb')
- **prom_type** – Type of the promise to instantiate.
- **kwargs** – kwargs of the promise initializer.

Return type *Promise*[~PromiseReturnType]

Returns

prompy.promio.jsonio module

Json related promise creators.

prompy.promio.jsonio.**dumps**(*data*, *prom_type=prompy.promise.Promise*, ***kwargs*)
Resolve the dumped data.

Parameters

- **data** (Union[dict, list]) –
- **prom_type** –
- **kwargs** –

Return type *Promise*[~PromiseReturnType]

Returns

prompy.promio.jsonio.**loads**(*data*, *prom_type=prompy.promise.Promise*, ***kwargs*)
Resolve the loaded data from a string.

Parameters

- **data** (str) –
- **prom_type** –
- **kwargs** –

Return type *Promise*[~PromiseReturnType]

Returns

prompy.promio.jsonio.**read_json_file**(*file*, *prom_type=prompy.promise.Promise*, ***kwargs*)
Resolve a json file content.

Parameters

- **file** (str) –
- **prom_type** –
- **kwargs** –

Return type *Promise*[~PromiseReturnType]

Returns

prompy.promio.jsonio.**write_json_file**(*file*, *content*, *prom_type=prompy.promise.Promise*, ***kwargs*)
Write the content to a json file. Resolve when done.

Parameters

- **file** (str) –
- **content** –
- **prom_type** –
- **kwargs** –

Return type *Promise*[~PromiseReturnType]

Returns

1.1.1.4 prompy.threadio package

prompy.threadio.pooled_caller module

class prompy.threadio.pooled_caller.**PooledCaller**(***pool_kwargs*)
Bases: *prompy.container.BasePromiseContainer*

Class wrapper for urlcall. Auto-add calls to a PromiseQueuePool to be resolved.

__init__(***pool_kwargs*)
Initialize self. See help(type(self)) for accurate signature.

add_promise(*promise*)
Add a promise to the container.

Parameters **promise** (*Promise*[~PromiseReturnType]) –

Returns

call(*url*, ***kwargs*)

Return type *Promise*[]

get(*url*, ***kwargs*)

head(*url*, ***kwargs*)

json_call(*url*, ***kwargs*)

post(*url*, ***kwargs*)

put(*url*, ***kwargs*)

prompy.threadio.promise_queue module

class prompy.threadio.promise_queue.**PromiseQueue**(*start=False*, *max_idle=0.5*, *on_stop=None*, *queue_timeout=0.01*, *interval=0.01*, *daemon=False*)
Bases: *prompy.container.PromiseContainer*

__init__(*start=False*, *max_idle=0.5*, *on_stop=None*, *queue_timeout=0.01*, *interval=0.01*, *daemon=False*)
Initialize self. See help(type(self)) for accurate signature.

add_promise(*promise*)
Add a promise to the container.

Parameters **promise** (*Promise*[~PromiseReturnType]) –

Returns

cancel(*cancel_id*)

error

running

start()

stop()

class prompy.threadio.promise_queue.**PromiseQueuePool**(*pool_size=8*, *start=False*, *max_idle=0.5*, *daemon=False*)
Bases: *prompy.container.BasePromiseRunner*

__init__(*pool_size=8*, *start=False*, *max_idle=0.5*, *daemon=False*)
Initialize self. See help(type(self)) for accurate signature.

add_promise(*promise*)
Add a promise to the container.

Parameters **promise** (*Promise*[~PromiseReturnType]) –

Returns

is_running()

on_thread_stop(*func*)

start()

stop()

prompy.threadio.tpromise module

Threaded Promise

Auto insert in a global thread pool.

Use the following environ vars:

- PROMPY_THREAD_POOL_SIZE=2
- PROMPY_THREAD_IDLE_TIME=0.5
- PROMPY_THREAD_DAEMON=false

class prompy.threadio.tpromise.**TPromise**(*starter*, **args*, ***kwargs*)
Bases: *prompy.promise.Promise*

A promise with auto insert in a threadio.PromiseQueue.

__init__(*starter*, **args*, ***kwargs*)
Promise takes at least a starter method with params to this promise resolve and reject. Does not call exec by default but with start_now the execution will be synchronous.

Parameters

- **starter** – otherwise known as executor.
- **then** – initial resolve callback
- **catch** – initial catch callback
- **complete** – initial complete callback
- **raise_again** – raise the rejection error again.
- **start_now** –
- **results_buffer_size** – number of results to keep in the buffer.

classmethod stop_queue()

classmethod wrap(*func*)

1.1.2 prompy.promise module

Promise for python

class prompy.promise.**Promise**(*starter*, *then=None*, *catch=None*, *complete=None*, *raise_again=False*, *start_now=False*, *results_buffer_size=100*)

Bases: typing.Generic

Promise interface Based on js Promises.

Basic usage:

p = Promise(lambda resolve, reject: resolve('Hello')).then(print)

__init__(*starter*, *then=None*, *catch=None*, *complete=None*, *raise_again=False*, *start_now=False*, *results_buffer_size=100*)

Promise takes at least a starter method with params to this promise resolve and reject. Does not call exec by default but with start_now the execution will be synchronous.

Parameters

- **starter** (Callable[[Callable, Callable], None]) – otherwise known as executor.
- **then** (Optional[Callable[[~PromiseReturnType], None]]) – initial resolve callback
- **catch** (Optional[Callable[[Exception], None]]) – initial catch callback
- **complete** (Optional[Callable[[Union[List[~PromiseReturnType], ~PromiseReturnType], Exception], None]]) – initial complete callback
- **raise_again** (bool) – raise the rejection error again.
- **start_now** (bool) –
- **results_buffer_size** (int) – number of results to keep in the buffer.

callback_handler(*obj*)

Override to handle the return value of callbacks.

Parameters **obj** (Any) – The return value of a callback

Returns

catch(*func*)

Add a callback to rejection

Parameters **func** (Callable[[Exception], None]) –

Returns

complete(*func*)

Add a callback to finally block

Parameters **func** (Callable[[Union[List[~PromiseReturnType], ~PromiseReturnType], Exception], None]) –

Returns

error

Return type Exception

exec()

Execute the starter method.

Returns

id

Return type UUID

reject(*error*)
Reject the promise.

Parameters **error** (Exception) –

Returns

resolve(*result*)
Resolve the promise, called by executor.

Parameters **result** (*~PromiseReturnType*) –

Returns

result

Return type Union[Tuple[~PromiseReturnType], ~PromiseReturnType]

state

Return type *PromiseState*

then(*func*)
Add a callback to resolve

Parameters **func** (Callable[[~PromiseReturnType], None]) – callback to resolve

Returns

class prompy.promise.**PromiseState**
Bases: enum.Enum

An enumeration.

fulfilled = 2

pending = 1

rejected = 3

1.1.3 prompy.awaitable module

Promise you can await.

Example

```
import asyncio

from prompy.awaitable import AwaitablePromise
from prompy.networkio.async_call import call

async def call_starter(resolve, _):
    google = await call('http://www.google.com')
    resolve(google)

p = AwaitablePromise(call_starter)

@p.then
```

(continues on next page)

(continued from previous page)

```
def then(result):
    print(result)
    asyncio.get_event_loop().stop()

@p.catch
def catch(err):
    asyncio.get_event_loop().stop()
    raise err

asyncio.get_event_loop().run_forever()
```

class prompy.awaitable.**AsyncPromiseRunner**

Bases: *prompy.container.BasePromiseRunner*

Run the loop forever

__init__()

Initialize self. See help(type(self)) for accurate signature.

add_promise(*promise*)

Add a promise to the container.

Parameters **promise** (*Promise*[~PromiseReturnType]) –

Returns

add_promises(**promises*)

Add all the promises.

Parameters **promises** (*Promise*[~PromiseReturnType]) – promises to add

Returns

start()

stop()

class prompy.awaitable.**AwaitablePromise**(*starter*, *then=None*, *catch=None*, *complete=None*, *loop=None*)

Bases: *prompy.promise.Promise*

asyncio compatible promise

Await it to get the result. Need a running loop to actually start the executor.

__init__(*starter*, *then=None*, *catch=None*, *complete=None*, *loop=None*)

Promise takes at least a starter method with params to this promise resolve and reject. Does not call exec by default but with start_now the execution will be synchronous.

Parameters

- **starter** (Callable[[Callable, Callable], None]) – otherwise known as executor.
- **then** (Optional[Callable[[~PromiseReturnType], None]]) – initial resolve callback
- **catch** (Optional[Callable[[Exception], None]]) – initial catch callback
- **complete** (Optional[Callable[[Union[List[~PromiseReturnType], ~PromiseReturnType], Exception], None]]) – initial complete callback
- **raise_again** – raise the rejection error again.

- **start_now** –
- **results_buffer_size** – number of results to keep in the buffer.

callback_handler(*obj*)
Override to handle the return value of callbacks.

Parameters **obj** (Any) – The return value of a callback

Returns

catch(*func*)
Add a callback to rejection

Parameters **func** (Callable[[Exception], None]) –

Returns

complete(*func*)
Add a callback to finally block

Parameters **func** (Callable[[Union[List[~PromiseReturnType], ~PromiseReturnType], Exception], None]) –

Returns

error

Raise invalid state if the promise was not completed.

Returns the exception or the handled error

exec()
Execute the starter method.

Returns

id

Return type UUID

reject(*error*)
Reject the promise.

Parameters **error** (Exception) –

Returns

resolve(*result*)
Resolve the promise, called by executor.

Parameters **result** (Any) –

Returns

result

Return type Union[Tuple[~PromiseReturnType], ~PromiseReturnType]

state

Return type *PromiseState*

then(*func*)
Add a callback to resolve

Parameters **func** (Callable[[~PromiseReturnType], None]) – callback to resolve

Returns

static wrap(*func*)

1.1.4 prompy.container module

class prompy.container.**BasePromiseContainer**
Bases: object

Interface for a promise container.

add_promise(*promise*)
Add a promise to the container.

Parameters **promise** (*Promise*[~PromiseReturnType]) –

Returns

add_promises(**promises*)
Add all the promises.

Parameters **promises** (*Promise*[~PromiseReturnType]) – promises to add

Returns

class prompy.container.**BasePromiseRunner**
Bases: *prompy.container.BasePromiseContainer*

A container that need to start and stop.

add_promise(*promise*)
Add a promise to the container.

Parameters **promise** (*Promise*[~PromiseReturnType]) –

Returns

start()

stop()

class prompy.container.**PromiseContainer**
Bases: *prompy.container.BasePromiseContainer*, collections.abc.Container

Basic promise container.

Keeps the promises in a dict with the promise id as key.

__init__()
Initialize self. See help(type(self)) for accurate signature.

add_promise(*promise*)
Add a promise to the container.

Parameters **promise** (*Promise*[~PromiseReturnType]) –

Returns

prompy.container.**container_wrap**(*func*)

Return type Callable[..., *Promise*[~PromiseReturnType]]

1.1.5 prompy.errors module

exception prompy.errors.**PromiseError**
Bases: Exception

Base promise error

exception prompy.errors.**PromiseRejectionError**
Bases: *prompy.errors.PromiseError*

Raised when a promise is called with raise_again option

exception prompy.errors.**UnhandledPromiseError**
Bases: *prompy.errors.PromiseError*

Unhandled promise rejection error

exception prompy.errors.**UrlCallError**
Bases: *prompy.errors.PromiseError*

Web call error

1.1.6 prompy.promtools module

Methods for working with promises.

prompy.promtools.**later**(*func*, *delay*, *wait_func=<built-in function sleep>*, *prom_type=prompy.promise.Promise*, ***kwargs*)
Bad, do not use.

Return type Callable[..., *Promise*[~PromiseReturnType]]

prompy.promtools.**pall**(**promises*, *prom_type=prompy.promise.Promise*, ***kwargs*)
Wrap all the promises in a single one that resolve when all promises are done.

Return type *Promise*[~PromiseReturnType]

prompy.promtools.**piter**(*func*, *iterable*, *prom_type=prompy.promise.Promise*, ***kwargs*)
Applies func to iterable in a promise, like a map and foreach (starter->then).

Return type *Promise*[~PromiseReturnType]

prompy.promtools.**promise_wrap**(*func*, *prom_type=prompy.promise.Promise*, ***kw*)
Wraps a function return in a promise resolve.

Return type Callable[..., *Promise*[~PromiseReturnType]]

CHAPTER 2

Prompy

Promises for python.

2.1 Installation

```
cd /path/to/install
git clone https://github.com/T4rk1n/prompy.git
pip install .
```

2.2 Usage

Create a promise

```
from prompy.promise import Promise

def promise_starter(resolve, reject):
    resolve('Hello')

promise = Promise(promise_starter)
promise.then(lambda result: print(result))

# Base promises run synchronously.
promise.exec()  # prints hello
```

2.3 Promise types

- AwaitablePromise - asyncio Promises you can await.
- TPromise - Promise that put itself in threaded queue pool.

- ProcessPromise - Promise to add to a process queue (manual insertion).

2.4 Url calls

Non-blocking promise wrappers for urllib requests.

Example:

```
from prompy.threadio.tpromise import TPromise
from prompy.networkio.urlcall import url_call

git = url_call('http://github.com', prom_type=TPromise)


@git.then
def gud(rep):
    print(rep.content)
```

2.5 Caller factory

Wraps a class methods starting with `call_` with a `url_call`.

Example using AwaitablePromise:

```
import asyncio

from prompy.networkio.call_factory import CallRoute, Caller
from prompy.awaitable import AwaitablePromise


class Api(Caller):
    def call_users(self, user_id, **kwargs):
        # methods with url params must have the same number of args
        return CallRoute('/users/<user_id>')

    def call_create_post(self, **kwargs):
        return CallRoute('/posts', method='POST')


api = Api(base_url='https://jsonplaceholder.typicode.com', prom_type=AwaitablePromise)


async def call_api():
    home = await api.call_users(1)
    print(home.content)
    data = await api.call_create_post(data={'title': 'foo', 'body': 'bar', 'userId':␣
↪3})
    print(data.content)

if __name__ == '__main__':
    asyncio.get_event_loop().run_until_complete(call_api())
```

Note: Since it use urllib, the asyncio loop will still block while waiting for the response.

2.6 Promise creators modules

- prompy.promio.fileio - Read, write, delete, compress, decompress and walk files.
- prompy.promio.jsonio - json encoding wrap.
- prompy.processio.proc - subprocess wrap.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

Symbols

A

B

C

R

S

T

U

W