hio

Release 0.3.4

Samuel M. Smith

Nov 27, 2022

CONTENTS:

CHAPTER

ONE

INTRODUCTION TO HIO

Weightless hierarchical asynchronous coroutines and I/O in Python.

Rich Flow Based Programming Hierarchical Structured Concurrency with Asynchronous IO

Hio builds on very early work on hierarchical structured concurrency with lifecycle contexts from [ioflo](https://ioflo.com), [ioflo github](https://github.com/ioflo/ioflo), and [ioflo manuals](https://github.com/ioflo/ioflo_manuals).

This approach is compatible with flow based programming that sees all components as asynchronous and linked by asynchronous buffers. FPB naturally lends itself to a much lighter weight async structure based on a hierarchical scheduling approach.

This is even lighter weight and more performant than non-hierarchical structured concurrency approaches such as trio or curio.

approach also is informed by and supports cooperative concurrent

discrete event simulation (DES). One important feature of concurrent discrete event simulation is reproducibility. This requires tight control over scheduling order as in completely deterministic control of scheduling. In order to have high fidelity reproduction or replay, all coroutines used in a discrete event simulation must be scheduled exactly in the same relative order. An asyncio event loop does not have such tight control over scheduling order. But Hio does and therefore can be used for discrete event simulations with high fidelity replay. One can always add noise and uncertainty to a Hio replay as needed, but due to its underlying deterministic scheduling even the addition of noise can be done in a predetermined reproducible way.

1.1 Structured Concurrency with Asynchronous IO

More recently the [curio](https://curio.readthedocs.io/en/latest/) and [trio](https://trio.readthedocs.io/en/stable/) libraries have popularized coroutine based [structured concurrency](https://en.wikipedia.org/wiki/Structured_concurrency).

See here for why it matters ... [here](https://vorpus.org/blog/notes-on-structured-concurrency-or-go-statement-considered-harmful/) and [here](https://vorpus.org/blog/companion-post-for-my-pycon-2018-talk-on-async-concurrency-using-trio/)

The main difference between hio and curio or trio is that hio uses extremely lightweight asynchronous hierarchical coroutine scheduling. The scheduler only does one thing, that is, time slice sub coroutines or sub coroutine schedulers.

The coroutines are responsible for managing the asynchronous IO not the scheduler. This is compatible with a flow based programming (FBP) approach where Async IO only services buffers. All interaction with other system components happens through those buffers not some other mechanism. And certainly not a mechanism provided by the async scheduler. This makes the architecture as flat as possible. All async IO is accessed via a buffer. Back pressure is naturally exhibited via the buffer state. This approach merges the best of FBP and a bare-bones coroutine based async.

See API docs on readthedocs.org [Here](https://hio-py.readthedocs.io/en/latest/index.html)

1.2 Current Status

Version 0.4.1

Refined Doist and DoDoer makes their protocol interfaces nearly identical as
as is reasonably practical

Added HTTP support with hio compatible HTTP Client and HTTP WSGI Server Example test code shows HTTP Server working with Falcon and Bottle ReST Micro

frameworks

Version 0.3.4

The async scheduler features should be pretty stable going forward. The tcp library should also be stable going forward.

The TCP IO Client and Server classes are implemented. Includes support for TLS

TCP ServerDoer, EchoServerDoer, and ClientDoer classes are implemented as examples

CHAPTER

TWO

SRC

2.1 hio package

2.1.1 Subpackages

hio.base package

Submodules

hio.base.basing module

hio.base.doing module

hio.base.tyming module

Module contents

hio.core package

Subpackages

hio.core.serial package

Submodules

hio.core.serial.serialing module

Module contents

hio.core.tcp package

Submodules

hio.core.tcp.clienting module

2.1.2 Submodules

2.1.3 hio.cli module

2.1.4 hio.daemon module

2.1.5 hio.hioing module

2.1.6 Module contents

CHAPTER

THREE

API REFERENCE

This page contains auto-generated API reference documentation[1].

3.1 hio

hio package

3.1.1 Subpackages

hio.base

hio.base Package

Submodules

hio.base.basing

hio.base.basing Module

Module Contents

hio.base.basing.**State**

hio.base.doing

hio.core.doing Module

[1] Created with sphinx-autoapi

Module Contents

Classes

Doist	Doist is the root coroutine scheduler
Doer	Doer base class for hierarchical structured async coroutine like generators.
ReDoer	ReDoer is an example sub class whose .recur is a generator method not a
DoDoer	DoDoer implements Doist like functionality to allow nested scheduling of Doers.
ExDoer	ExDoer is example Doer for testing and demonstration
TryDoer	TryDoer supports testing with methods to record sends and yields

Functions

doify(f, *[, name, tock])	Returns Doist compatible copy, g, of converted generator function f.
doize(*[, tock])	Returns decorator that makes decorated generator function Doist compatible.
bareDo([tymth, tock])	Bare bones generator function template as example of generator function
doifyExDo(tymth[, tock, states])	Example generator function for testing and demonstration.
doizeExDo(tymth[, tock, states])	Example decorated generator function for use with doize decorator.
tryDo(states, tymth[, tock])	Generator function test example non-class based generator.

Attributes

Deed

hio.base.doing.**Deed**

class hio.base.doing.**Doist**(*real=False*, *limit=None*, *doers=None*, ***kwa*)

Bases: *hio.base.tyming.Tymist*

Doist is the root coroutine scheduler Provides relative cycle time in seconds with .tyme property to doers it runs The relative cycle time is advanced in .tock size increments by the by the .tick method. The doist may treat .tyme as artificial time or synchonize it to real time.

.enter method prepares deeds deque of triples (dog, retyme, doer) where
dog is a doer generator returned by calling doer generator instances, functions, or methods.

.recur method runs its deeds deque of triples (dog, retyme, doer) once per
invocation. This synchronizes their cycle time .tyme to the Doist's tyme.

.do method repeatedly runs .recur until generators are complete
it may either repeat as fast as possbile or repeat at real time increments.

Inherited Class Attributes:
.Tock is default .tock

`real`

True means run in real time, Otherwise as fast as possible.

Type
boolean

`limit`

maximum run tyme limit then closes all doers

Type
float

`done`

True means completed due to limit or all deeds completed False is forced complete due to error

Type
boolean

`doers`

Doer class instances, generator methods or function callables with attributes tock, done, and opts dict(). Used throughout the execution lifecycle.

Type
list

`deeds`

Tuples of form (dog, retyme, doer). Where: dog is generator created by doer retyme is tyme (real or simulated) in seconds when dog should run next doer is associated doer in .doers list used to assign its .done state

given completion state of its dog

Used throughout the execution lifecycle. The normal case is use the default empty initialization performed here and update in .enter().

Type
deque

`timer`

for real time intervals

Type
MonoTimer

Inherited Properties:
tyme: is float relative cycle time, .tyme is artificial time : is float tyme increment of .tick()

Properties:

Inherited Methods:
.tick increments .tyme by one .tock or provided tock

`.enter prepare deeds, deque of triples (dog, retyme, doer)`

.recur run through all deeds once

.do repeadedly call .recur until all dogs in deeds are complete or
times out do to reaching time limit

do(*doers=None*, *limit=None*, *tyme=None*)

Readies deeds deque from .doers or doers if any and then iteratively runs .recur over deeds deque until completion of all deeds. Each entry in deeds is a triple (dog, retyme, doer) where:

dog is generator retyme is tyme (real or simulated) in seconds when dog should run next doer is from .doers list used to assign its .done state given associated completion state of its dog

If interrupted by exception call .close on each dog to force exit context.

Keyboard interrupt (cntl-c) forces exit.

Once finally clause closes a generator it must be reinited before it can be run again

Parameters

- **doers** (*iterable*) – generator method or function callables with attributes tock, done, and opts dict(). This may be used to update the .doers attribute which is used throughout the execution lifecycle. If not provided uses .doers. Parameterization here of doers enables some special cases. The normal case is to initialize in .__init__ or here.
- **limit** (*float*) – is real time limit on execution. Forces close of all dogs.
- **tyme** (*float*) – is optional starting tyme. Resets .tyme to tyme whe provided. If not provided uses current .tyme

Returns
None

See: https://stackoverflow.com/questions/40528867/setting-attributes-on-func For setting attributes on bound methods.

enter(*doers=None*)

Enter context Returns (deque): deeds deque of triples (dog, retyme, doer) where:

dog is generator retyme is tyme (real or simulated) in seconds when dog should run next doer is from .doers list used to assign its .done state given

completion state of its dog

Calls each generator callable (instance or function or method) in .doers to create each generator dog. Injects own tymth function closure, and

generator function's own tock, and opts.

Runs enter context of each dog by calling next(dog)

Parameters

- **attributes** (*doers is list of generator method or function callables with*) – .tock is tyme increment in seconds .done is Boolean completion state .opts is dict() of optional parameters If not provided uses .doers. The normal case is to initialize in .__init__. or .do().
- **triples** (*deeds is deque of deed*) –

Returns
A deed is tuple of form (dog, retyme, doer). If not provided uses .deeds.

Return type
deeds deque()

See: https://stackoverflow.com/questions/40528867/setting-attributes-on-func For setting attributes on bound methods.

recur(*deeds=None*)

Recur once through deeds deque of tuples (triples) of form (dog, retyme, doer) and update in place

Each deed is deque of tuples of form (dog, retyme, doer) where:
dog is generator retyme is tyme (real or simulated) in seconds when dog should run next doer is from .doers list used to assign its .done state given associated completion state of its dog

Each cycle checks all generators in deeds deque and runs if retyme past. At end of cycle advances .tyme by one .tock by calling .tick()

Parameters
deeds (*deque*) – tuples of form (dog, retyme, doer). Parameterization here of deeds enables some special cases.

The Parameterization here of deeds enables some special cases such as manual testing or iteraton. The normal case is to initialize .doers in .__init__. or .do() and to initialize .deeds in .__init__. and then update in .enter()

exit(*deeds=None*)

Force exit each still opened deed calling .close on the dog generator which throws a GeneratorExit to the generator. This executes the close context (GeneratorExit) which then excecutes the exit context in the finally caluse. Each dogs exit is responsible for releasing resources Previously aborted or closed dogs have already exited Close any running dogs in reverse order so that enters and exits are nested pairs so that the corresponding exits appear in reverse order to their entes. This preserves nested resource dependencies. For example:

enter A,

enter B,
enter C, exit C,

exit B,

exit A

Parameters
deeds (*deque*) – tuples of form (dog, retyme, doer). If not provided uses .deeds. Parameterization here of deeds enables some special cases.

extend(*doers*)

Extend .doers list with doers. Ready deeds from doers and extend .doers and .deeds. Edit deeds in place so not replace deque.

Parameters
extension. (*doers is list of doers to add as*) –

remove(*doers*)

Remove doers from .doers list and any associated deeds from .deeds deque. Force close removed deeds.

Parameters
remove. (*doers is list of doers to*) –

hio.base.doing.**doify**(*f*, *, *name=None*, *tock=0.0*, ***opts*)

Returns Doist compatible copy, g, of converted generator function f. Each invoction of doify(f) returns a unique copy of doified function f. Imbues copy, g, of converted generator function, f, with attributes used by Doist.enter() or DoDoer.enter(). Allows multiple instances of copy, g, of generator function, f, each with unique attributes.

Usage: def f():

pass

c = doify(f, name='c')

Parameters

- **function** (*f is generator*) –
- **copy** (*name is new function name for returned doified copy g. Default is to*) – f.__name__
- **g** (*tock is default tock attribute of doified copy*) –
- **attribute** (*opts is dictionary of remaining parameters that becomes .opts*) – of doified copy g

Based on: https://stackoverflow.com/questions/972/adding-a-method-to-an-existing-object-instance

hio.base.doing.**doize**(*, *tock=0.0*, **opts*)

Returns decorator that makes decorated generator function Doist compatible. Imbues decorated generator function with attributes used by Doist.enter() or DoDoer.enter(). Only one instance of decorated function with shared attributes is allowed.

Usage: @doize def f():

pass

Parameters

- **f** (*tock is default tock attribute of doized*) –
- **attribute** (*opts is dictionary of remaining parameters that becomes .opts*) – of doized f

class hio.base.doing.**Doer**(*, *tymth=None*, *tock=0.0*, **opts*)

Bases: *hio.base.tyming.Tymee*

Doer base class for hierarchical structured async coroutine like generators. Doer.__call__ on instance returns generator. Interface for Doist etc is generator function like object. Doer is generator method instance creator and has extra methods and attributes that a plain generator function does not

The .do method executes other methods each corresponding to one of the six econtexts:

enter, recur, clean, exit, (unforced) close, abort (forced)

Actual context order may be one of:
enter, recur, clean, exit enter, recur, close, exit enter, recur, abort, exit enter, abort, exit

.done is Boolean completion state

True means completed Otherwise incomplete. Incompletion maybe due to close or abort.

.opts is dict of injected options into its .do generator by scheduler

Inherited Properties:

.tyme is float relative cycle time of associated Tymist .tyme obtained
via injected .tymth function wrapper closure.

.tymth is function wrapper closure returned by Tymist .tymeth() method.
When .tymth is called it returns associated Tymist .tyme. .tymth provides injected dependency on Tymist tyme base.

Properties:

.tock is float, desired time in seconds between runs or until next run,
non negative, zero means run asap

Inherited Methods:
.wind injects ._tymth dependency from associated Tymist to get its .tyme

.__call__ makes instance callable
Appears as generator function that returns generator

.do is generator method that returns generator

.enter is enter context action method

.recur is recur context action method or generator method

.clean is clean context action method

.exit is exit context method

.close is close context method

.abort is abort context method

Hidden:

._tymth is injected function wrapper closure returned by .tymen() of
associated Tymist instance that returns Tymist .tyme. when called.

._tock is hidden attribute for .tock property

property tock
tock property getter, get ._tock .tock is float desired .tyme increment in seconds

__call__(***kwa*)
Returns generator Does not advance to first yield. The advance to first yield effectively invodes the enter or open context on the generator. To enter either call .next or .send(None) on generator

do(*tymth*, *, *tock=0.0*, ***opts*)
Generator method to run this doer. Calling this method returns generator. Interface matches generator function for compatibility. To customize create subclasses and override the lifecycle methods:

.enter, .recur, .exit, .close, .abort

Parameters

- **of**(*tymth is injected function wrapper closure returned by .tymen()*) – Tymist instance. Calling tymth() returns associated Tymist .tyme.
- **value**(*tock is injected initial tock*) –
- **parameters**(*args is dict of injected optional additional*) –

enter()
Do 'enter' context actions. Override in subclass. Not a generator method. Set up resources. Comparable to context manager enter.

recur(*tyme*)

Do 'recur' context actions. Override in subclass. Regular method that perform repetitive actions once per invocation. Assumes resource setup in .enter() and resource takedown in .exit() (see ReDoer below for example of .recur that is a generator method)

Returns completion state of recurrence actions.
True means done False means continue

Parameters
here. (*Doist feeds its .tyme through .send to .do yield which passes it*) –

.recur maybe implemented by a subclass either as a non-generator method or a generator method. This stub here is as a non-generator method. The base class .do detects which type:

If non-generator .do method runs .recur method once per iteration
until .recur returns (True)

If generator .do method runs .recur with (yield from) until .recur
returns (see ReDoer for example of generator .recur)

clean()

Do 'clean' context actions. Override in subclass. Not a generator method. Clean up resources that are unique to a clean exit. Called by else after normal return.

exit()

Do 'exit' context actions. Override in subclass. Not a generator method. Clean up resources. Comparable to context manager exit. Called by finally after normal return, close, or abort. After .exit() do returns resulting in StopIteration.

close()

Do 'close' context actions. Override in subclass. Not a generator method. Forced close by thrown generator .close() method causing GeneratorExit. .exit() is finally called after .close().

abort(*ex*)

Do 'abort' context actions. Override in subclass. Not a generator method. :param ex is Exception instance that caused abort.:

Unexpected exception that results in generator exiting but not GeneratorExit. .exit() is finally called after .abort().

class hio.base.doing.**ReDoer**(*, *tymth=None*, *tock=0.0*, ***opts*)

Bases: *Doer*

ReDoer is an example sub class whose .recur is a generator method not a plain method. Its .do method detects that its .recur is a generator method and executes it using yield from instead of just calling the method.

Inherited Attributes:

.done is Boolean completion state:
True means completed Otherwise incomplete. Incompletion maybe due to close or abort.

.opts is dict of injected options into its .do generator by scheduler

Inherited Properties:

.tyme is float relative cycle time of associated Tymist .tyme obtained
via injected .tymth function wrapper closure.

.tymth is function wrapper closure returned by Tymist .tymeth() method.
When .tymth is called it returns associated Tymist .tyme. .tymth provides injected dependency on Tymist tyme base.

.tock is float, desired time in seconds between runs or until next run,
non negative, zero means run asap

Inherited Methods:
.wind injects ._tymth dependency from associated Tymist to get its .tyme .__call__ makes instance callable

Appears as generator function that returns generator

.do is generator method that returns generator .enter is enter context action method .recur is recur context action method or generator method .exit is exit context method .close is close context method .abort is abort context method

Overidden Methods:
.recur

Hidden:

._tymth is injected function wrapper closure returned by .tymen() of
associated Tymist instance that returns Tymist .tyme. when called.

._tock is hidden attribute for .tock property

`recur()`

Do 'recur' context actions as a generator method. Override in subclass. Assumes resource setup in .enter() and resource takedown in .exit() (see Doer for example of .recur that is a regular method)

yield the current .tock accepts the current tyme returns the .done

Parameters

- **yield** (*tyme is initial output of send fed to do*) –
- **.tyme** (*Doist feeds its*) –

Returns completion state of recurrence actions.
True means done False means continue

Maybe a non-generator method or a generator method. For base class do:

non-generator recur method runs until returns (True) generator recur method runs until returns (yield from)

`class hio.base.doing.DoDoer(doers=None, always=False, **kwa)`

Bases: *Doer*

DoDoer implements Doist like functionality to allow nested scheduling of Doers. Each DoDoer runs a list of doers like a Doist but using the tyme from its

injected tymth for the associated tymist as injected by its ultimate root parent Doist and any intervening parent DoDoer(s).

Scheduling hierarchy: Doist->DoDoer...->DoDoer->Doers

Inherited Attributes:

.done is Boolean completion state:
True means completed Otherwise incomplete. Incompletion maybe due to close or abort.

.opts is dict of injected options for its generator .do

Attributes:

Inherited Properties:

.tyme is float relative cycle time of associated Tymist .tyme obtained
via injected .tymth function wrapper closure.

.tymth is function wrapper closure returned by Tymist .tymeth() method.
When .tymth is called it returns associated Tymist .tyme. .tymth provides injected dependency on Tymist tyme base.

.tock is float, desired time in seconds between runs or until next run,
non negative, zero means run asap

Properties:

doers (list): Doer or Doist compatible generator instances,
functions, or methods.

deeds (deque): tuples of form (dog, retyme, doer) where:
dog is generator created by doer. retyme is tyme in seconds when next should run may be real or simulated. doer is associated doer in .doers list. Used throughout the execution lifecycle. The normal case is use the default empty initialization performed here and update in .enter().

always (bool): True means keep running even when all dogs in deeds
are complete. Enables dynamically managing extending or removing doers and associated deeds while running.

Inherited Methods:
.wind injects ._tymth dependency from associated Tymist to get its .tyme .__call__ makes instance callable

Appears as generator function that returns generator

.do is generator method that returns generator .enter is enter context action method .recur is recur context action method or generator method .clean is clean context action method .exit is exit context method .close is close context method .abort is abort context method

Overidden Methods:
.do .enter .recur .exit

Hidden:

._tymth is injected function wrapper closure returned by .tymen() of
associated Tymist instance that returns Tymist .tyme. when called.

._tock is hidden attribute for .tock property ._always is hidden attribute for .always property ._doers is hidden attribute for .doers property ._deeds is hidden attribute for .deeds property

`property doers`

doers property getter, get ._doers .doers is list of doist compatible generator instances, functions, or

methods

`property deeds`

deeds property getter, get ._deeds .deeds is deque of triples, each of form (dog, retyme, doer)

`property always`

always property getter, get ._always .always is Boolean, True means keep running even when all dogs in deeds

are complete. Enables dynamically managing extending or removing doers and associated deeds while running.

do(*tymth*, *tock=0.0*, *doers=None*, *always=None*, ***opts*)

Generator method to run this doer. Equivalent of doist.do Calling this method returns generator Interface matched generator function for compatibility

Parameters

- **of** (*tymth is injected function wrapper closure returned by .tymen()*) – Tymist instance. Calling tymth() returns associated Tymist .tyme.
- **value** (*tock is injected initial tock*) –
- **attributes** (*doers is list of generator method or function callables with*) – tock, done, and opts dict(). This may be used to update the .doers attribute which is used throughout the execution lifecycle. If not provided uses .doers. Parameterization here of doers enables some special cases. The normal case is to initialize in .__init__.
- **deeds** (*always is Boolean. True means keep running even when all dogs in*) – are complete. Enables dynamically managing extending or removing doers and associated deeds while running. When not provided use .always.
- **parameters** (*opts is dict of injected optional additional*) –

enter(*doers=None*)

Do 'enter' context actions. Equivalent of Doist.enter()

Returns deeds deque of triples (dog, retyme, doer) where:
dog is generator created by doer retyme is tyme in seconds when next should run may be real or simulated doer is doer for dog from doers list

Calls each generator callable (function or method) in .doers to create each generator dog.

Runs enter context of each dog by calling next(dog)

Parameters
doers (*list*) – Doer Instance, generator method or function callables with attributes tock, done, and opts dict(). If not provided uses .doers. Parameterization here of doers enables some special cases. The normal case is to initialize in .__init__.

Returns
A deed is tuple of form (dog, retyme, doer). If not provided uses .deeds.

Return type
deeds deque()

See: https://stackoverflow.com/questions/40528867/setting-attributes-on-func For setting attributes on bound methods.

recur(*tyme*, *deeds=None*)

Do 'recur' context actions. Equivalent of Doist.recur

Parameters
tyme (*float*) –

is output of send fed to do yield, The root scheduler
Doist feeds its .tyme which propagates down the chain of DoDoers Because tymist is injected by doist or dodoer, self.tyme is same as tyme. So may use either which is more convenient.

deeds (deque): tuples of form (dog, retyme, doer).
If not provided uses .deeds. Parameterization here of deeds enables some special cases.

Returns completion state of recurrence actions.
True means done False means continue

Cycle once through deeds deque and update in place

Each cycle checks all generators dogs in deeds deque and runs if retyme past.

exit(*deeds=None*)

Do 'exit' context actions.

Parameters
deeds (*deque*) – of deed tuples of form (dog, retyme, doer) If not provided uses .deeds. Parameterization here of deeds enables some special cases.

See: https://stackoverflow.com/questions/40528867/setting-attributes-on-func For setting attributes on bound methods.

extend(*doers*)

Extend .doers list with doers. Ready deeds from doers and extend .doers and .deeds. Edit deeds in place so not replace deque.

Parameters
extension. (*doers is list of doers to add as*) –

remove(*doers*)

Remove doers from .doers list and any associated deeds from .deeds deque. Force close removed deeds.

Parameters
remove. (*doers is list of doers to*) –

hio.base.doing.**bareDo**(*tymth=None*, *tock=0.0*, ***opts*)

Bare bones generator function template as example of generator function suitable for use with either doify wrapper or doize decorator. Make copy and rename for given application. Calling copied renamed function returns basic generator. Wrapping copied renamed function with doify returns yet unique wrapped copy with unique values of injected attributes and parameters and further renamed by wrapper. Decorating copied renamed function with doize returns singleton with injected parameter values.

Injected Attributes:
g.tock = tock # default tock attributes g.done = None # default done state g.opts

Parameters

- **of** (*tymth is injected function wrapper closure returned by .tymen()*) – Tymist instance. Calling tymth() returns associated Tymist .tyme.
- **value** (*tock is injected initial tock*) –
- **parameters** (*opts is dict of injected optional additional*) –

The function comments show where the 6 equivalent contexts are performed enter, recur, clean, exit, (unforced) close, abort (forced) So context order may be: enter, recur, clean, exit enter, recur, close, exit enter, recur, abort, exit enter, abort, exit

class hio.base.doing.**ExDoer**(***kwa*)

Bases: *Doer*

ExDoer is example Doer for testing and demonstration Supports introspection with methods to record sends and yields

See Doer for inherited attributes, properties, and methods.

.states is list of State namedtuples

Type
tyme, context, feed, count

.count is iteration count

enter()

recur(*tyme*)

exit()

close()

abort(*ex*)

hio.base.doing.**doifyExDo**(*tymth*, *tock=0.0*, *states=None*, ***opts*)

Example generator function for testing and demonstration. Example non-class based generator for use with doify wrapper. Calling this function returns generator. Wrapping this function with doify returns copy with unique attributes

Parameters

- **of** (*tymth is injected function wrapper closure returned by .tymen()*) – Tymist instance. Calling tymth() returns associated Tymist .tyme.
- **value** (*tock is injected initial tock*) –
- **namedtuples** (*states is list of State*) –
- **parameters** (*opts is dict of injected optional additional*) –

hio.base.doing.**doizeExDo**(*tymth*, *tock=0.0*, *states=None*, ***opts*)

Example decorated generator function for use with doize decorator. Example non-class based generator Calling this function returns generator

Parameters
of (*tymth is injected function wrapper closure returned by .tymen()*) – Tymist instance. Calling tymth() returns associated Tymist .tyme. tock is injected initial tock value states is list of State namedtuples (tyme, context, feed, count) opts is dict of injected optional additional parameters

class hio.base.doing.**TryDoer**(*stop=3*, ***kwa*)

Bases: *Doer*

TryDoer supports testing with methods to record sends and yields

Inherited Attributes:

.done is Boolean completion state:
True means completed Otherwise incomplete. Incompletion maybe due to close or abort.

.states is list of State namedtuples

Type
tyme, context, feed, count

.count is context count

.stop is stop count where doer completes

Inherited Properties:

.tyme is float relative cycle time of associated Tymist .tyme obtained
via injected .tymth function wrapper closure.

.tymth is function wrapper closure returned by Tymist .tymeth() method.
When .tymth is called it returns associated Tymist .tyme. .tymth provides injected dependency on Tymist tyme base.

.tock is float, desired time in seconds between runs or until next run,
non negative, zero means run asap

Properties:

.wind injects ._tymth dependency from associated Tymist to get its .tyme

.__call__ makes instance callable
Appears as generator function that returns generator

.do is generator method that returns generator

.enter is enter context action method

.recur is recur context action method or generator method

.exit is exit context method

.close is close context method

.abort is abort context method

enter()

recur(*tyme*)

exit()

close()

abort(*ex*)

hio.base.doing.**tryDo**(*states*, *tymth*, *tock=0.0*, ***opts*)
Generator function test example non-class based generator. Calling this function returns generator

hio.base.filing

hio.base.filing module

Module Contents

Classes

Filer	Filer instances manage file directories and files to hold keri installation
FilerDoer	Basic Filer Doer

Functions

openFiler([cls, name, temp, reopen, clear])	Context manager wrapper Filer instances for managing a filesystem directory

Attributes

logger	

hio.base.filing.**logger**

hio.base.filing.**openFiler**(*cls=None*, *name='test'*, *temp=True*, *reopen=True*, *clear=False*, ***kwa*)

Context manager wrapper Filer instances for managing a filesystem directory and or files in a directory.

Defaults to using temporary directory path. Context 'with' statements call .close on exit of 'with' block

Parameters

- **instance** (*cls is Class instance of subclass*) –
- **oglers** (*name is str name of ogler instance for filename so can have multiple*) – at different paths thar each use different log file directories
- **Boolean** (*temp is*) – Otherwise open in persistent directory, do not clear on close
- **directory** (*True means open in temporary*) – Otherwise open in persistent directory, do not clear on close
- **close** (*clear on*) – Otherwise open in persistent directory, do not clear on close

Usage:

with openFiler(name="bob") as filer:

with openFiler(name="eve", cls=FilerSubClass) as filer:

class hio.base.filing.**Filer**(*name='main'*, *base=''*, *temp=False*, *headDirPath=None*, *perm=None*, *reopen=True*, *clear=False*, *reuse=False*, *clean=False*, *filed=False*, *mode=None*, *fext=None*, ***kwa*)

Filer instances manage file directories and files to hold keri installation specific resources like databases and configuration files.

name

unique path component used in directory or file path name

Type
str

base

another unique path component inserted before name

Type
str

temp

True means use /tmp directory

Type
bool

headDirPath is head directory path

path is full directory path

perm is numeric os permissions for directory and/or file

Type
s

filed

True means .path ends in file. False means .path ends in directory

Type
bool

mode

file open mode if filed

Type
str

fext

file extension if filed

Type
str

file

Type
File

opened is Boolean, True means directory created and if file then file

is opened. False otherwise

File/Directory Creation Mode Notes:

.Perm provides default restricted access permissions to directory and/or files stat.S_ISVTX | stat.S_IRUSR | stat.S_IWUSR | stat.S_IXUSR 0o1700==960

stat.S_ISVTX is Sticky bit. When this bit is set on a directory it means

that a file in that directory can be renamed or deleted only by the owner of the file, by the owner of the directory, or by a privileged process. When this bit is set on a file it means nothing

stat.S_IRUSR Owner has read permission. stat.S_IWUSR Owner has write permission. stat.S_IXUSR Owner has execute permission.

HeadDirPath = /usr/local/var

TailDirPath = hio

CleanTailDirPath = hio/clean

AltHeadDirPath = ~

AltTailDirPath = .hio

AltCleanTailDirPath = .hio/clean

TempHeadDir = /tmp

TempPrefix = hio_

TempSuffix = _test

Perm

Mode = r+

Fext = text

reopen(*temp=None*, *headDirPath=None*, *perm=None*, *clear=False*, *reuse=False*, *clean=False*, *mode=None*, *fext=None*, ***kwa*)

Open if closed or close and reopen if opened or create and open if not

Parameters

- **temp** (*bool*) – assign to .temp True means open in temporary directory, clear on close False means open persistent directory, do not clear on close
- **headDirPath** (*str*) – optional head directory pathname for main database Default .HeadDirpath
- **perm** (*int*) – optional numeric os dir permissions for database directory and database files. Default .Perm
- **clear** (*bool*) – True means remove directory upon close False means do not remove directory upon close
- **reuse** (*bool*) – True means reuse self.path if already exists False means do not reuse but remake self.path
- **clean** (*bool*) – True means path uses clean tail variant False means path uses normal tail variant
- **mode** (*str*) – file open mode when .filed
- **fext** (*str*) – File extension when .filed

remake(***, *name=''*, *base=''*, *temp=None*, *headDirPath=None*, *perm=None*, *clean=False*, *filed=False*, *mode=None*, *fext=None*, ***kwa*)

Make and return (path. file) by opening or creating and opening if not preexistent, directory or file at path

Parameters

- **name** (*str*) – unique name alias portion of path

- **base** (*str*) – optional base inserted before name in path
- **temp** (*bool*) – optional None means ignore, True means open temporary directory, may clear on close False menans open persistent directory, may not clear on close
- **headDirPath** (*str*) – optional head directory pathname of main database
- **perm** (*int*) – directory or file permissions such as stat.S_IRUSR Owner has read permission. stat.S_IWUSR Owner has write permission. stat.S_IXUSR Owner has execute permission.
- **clean** (*bool*) – True means make path for cleaned version and remove old directory or file at clean path if any. False means make path normally (not clean)
- **filed** (*bool*) – True means .path is file path not directory path False means .path is directiory path not file path
- **mode** (*str*) – file open mode when .filed such as "w+"
- **fext** (*str*) – File extension when .filed

close(*clear=False*)

Close .file if any and if clear rm directory or file at .path

Parameters
clear (*bool*) – True means remove dir or file at .path

_clearPath()

Remove directory/file at end of .path

class hio.base.filing.**FilerDoer**(*filer*, ***kwa*)

Bases: *hio.base.doing.Doer*

Basic Filer Doer

done

completion state: True means completed Otherwise incomplete. Incompletion maybe due to close or abort.

Type
bool

filer

instance

Type
Filer

Properties:

tyme (float): relative cycle time of associated Tymist .tyme obtained
via injected .tymth function wrapper closure.

tymth (func): closure returned by Tymist .tymeth() method.
When .tymth is called it returns associated Tymist .tyme. .tymth provides injected dependency on Tymist tyme base.

tock (float)): desired time in seconds between runs or until next run,
non negative, zero means run asap

enter()

exit()

hio.base.tyming

hio.core.tyming Module

Module Contents

Classes

Tymist	Tymist keeps artificial or simulated or cycle time, called tyme.
Tymee	Tymee has .tyme property that returns the artificial or simulated or cycle time
Tymer	Tymer class to measure cycle time given by .tyme property of Tymist instance.

class hio.base.tyming.**Tymist**(*tyme=0.0*, *tock=None*, ***kwa*)

Bases: *hio.hioing.Mixin*

Tymist keeps artificial or simulated or cycle time, called tyme. Provides relative cycle time, tyme, in seconds with .tyme property in incremets of .tock seconds. .tyme is advanced one .tock increment with .tick method. .tyme may be synchronized with real time by a .tyme manager

Class Attributes:
.Tock is default .tock

Attributes:

Properties:
.tyme is float relative cycle time, .tyme is artificial time .tock is float tyme increment of .tick()

.tick increments .tyme by one .tock or provided tock

property tyme

tyme property getter, get ._tyme .tyme is float cycle time in seconds

property tock

tock property getter, get ._tock .tock is float cycle time .tyme increment in seconds

Tock = 0.03125

tick(*tock=None*)

Advance cycle time .tyme by tock seconds when provided othewise by .tock and return new .tyme :param tock is float of amount of time in seconds to change .tyme:

tymen()

Returns function wrapper closure tymth, when called tymth() returns .tyme. This enables read only injection of .tyme into any object via tymth() that wants to be on or access this Tymist's tyme base.

class hio.base.tyming.**Tymee**(*tymth=None*, ***kwa*)

Bases: *hio.hioing.Mixin*

Tymee has .tyme property that returns the artificial or simulated or cycle time from its referenced Tymist instance ._tymist.

Attributes:

Properties:

.tyme is float relative cycle time of associated Tymist .tyme obtained
via injected .tymth function wrapper closure.

.tymth is function wrapper closure returned by Tymist .tymeth() method.
When .tymth is called it returns associated Tymist .tyme. .tymth provides injected dependency on Tymist tyme base.

`.wind  injects ._tymth dependency from associated Tymist to get its .tyme`

Hidden:

._tymth is injected function wrapper closure returned by .tymen() of
associated Tymist instance that returns Tymist .tyme. when called.

`property tyme`

tyme property getter, get ._tyme .tyme is float cycle time in seconds

`property tymth`

tymth property getter, get ._tymth returns own copy of tymist.tynth function wrapper closure for subsequent injection into related objects that want to be on same tymist tyme base.

`wind`(*tymth*)

Inject new tymist.tymth as new ._tymth. Changes tymist.tyme base. Override in subclasses to update any dependencies on a change in tymist.tymth base

`class hio.base.tyming.Tymer`(*duration=None*, *start=None*, ***kwa*)

Bases: *Tymee*

Tymer class to measure cycle time given by .tyme property of Tymist instance. tyme is relative cycle time either artificial or real

Inherited Attributes

Attributes:

Inherited Properties:

.tyme is float relative cycle time of associated Tymist .tyme obtained
via injected .tymth function wrapper closure.

.tymth is function wrapper closure returned by Tymist .tymeth() method.
When .tymth is called it returns associated Tymist .tyme. .tymth provides injected dependency on Tymist tyme base.

Properties:
.duration = tyme duration of tymer in seconds from ._start to ._stop .elaspsed = tyme elasped in seconds since ._start .remaining = tyme remaining in seconds until ._stop .expired = True if expired, False otherwise, i.e. .tyme >= ._stop

Inherited Methods:
.wind is injects ._tymth dependency

`.start() = start tymer at current .tyme`

`.restart() = restart tymer at last ._stop so no time lost`

Hidden:

._tymth is injected function wrapper closure returned by .tymen() of
associated Tymist instance that returns Tymist .tyme. when called.

._start is start tyme in seconds ._stop is stop tyme in seconds

property duration

duration property getter, .duration = ._stop - ._start .duration is float duration tyme

property elapsed

elapsed tyme property getter, Returns elapsed tyme in seconds (fractional) since ._start.

property remaining

remaining tyme property getter, Returns remaining tyme in seconds (fractional) before ._stop.

property expired

Returns True if tymer has expired, False otherwise. .tyme >= ._stop,

Duration = 0.0

wind(*tymth*)

Inject new ._tymist and any other bundled tymee references Update any dependencies on a change in ._tymist:

starts over itself at new ._tymists time

start(*duration=None*, *start=None*)

Starts Tymer of duration secs at start time start secs.
If duration not provided then uses current duration If start not provided then starts at current .tyme

restart(*duration=None*)

Lossless restart of Tymer at .tyme = ._stop for duration if provided, current duration otherwise No time lost. Useful to extend Tymer so no time lost

Package Contents

Classes

Tymist	Tymist keeps artificial or simulated or cycle time, called tyme.
Tymee	Tymee has .tyme property that returns the artificial or simulated or cycle time
Tymer	Tymer class to measure cycle time given by .tyme property of Tymist instance.
Doist	Doist is the root coroutine scheduler
Doer	Doer base class for hierarchical structured async coroutine like generators.
DoDoer	DoDoer implements Doist like functionality to allow nested scheduling of Doers.
Filer	Filer instances manage file directories and files to hold keri installation
FilerDoer	Basic Filer Doer

Functions

doize(*[, tock])	Returns decorator that makes decorated generator function Doist compatible.
doify(f, *[, name, tock])	Returns Doist compatible copy, g, of converted generator function f.
openFiler([cls, name, temp, reopen, clear])	Context manager wrapper Filer instances for managing a filesystem directory

class hio.base.**Tymist**(*tyme=0.0*, *tock=None*, ***kwa*)

Bases: hio.hioing.Mixin

Tymist keeps artificial or simulated or cycle time, called tyme. Provides relative cycle time, tyme, in seconds with .tyme property in incremets of .tock seconds. .tyme is advanced one .tock increment with .tick method. .tyme may be synchronized with real time by a .tyme manager

Class Attributes:
.Tock is default .tock

Attributes:

Properties:
.tyme is float relative cycle time, .tyme is artificial time .tock is float tyme increment of .tick()

.tick increments .tyme by one .tock or provided tock

property tyme

tyme property getter, get ._tyme .tyme is float cycle time in seconds

property tock

tock property getter, get ._tock .tock is float cycle time .tyme increment in seconds

Tock = 0.03125

tick(*tock=None*)

Advance cycle time .tyme by tock seconds when provided othewise by .tock and return new .tyme :param tock is float of amount of time in seconds to change .tyme:

tymen()

Returns function wrapper closure tymth, when called tymth() returns .tyme. This enables read only injection of .tyme into any object via tymth() that wants to be on or access this Tymist's tyme base.

class hio.base.**Tymee**(*tymth=None*, ***kwa*)

Bases: hio.hioing.Mixin

Tymee has .tyme property that returns the artificial or simulated or cycle time from its referenced Tymist instance ._tymist.

Attributes:

Properties:

.tyme is float relative cycle time of associated Tymist .tyme obtained
via injected .tymth function wrapper closure.

.tymth is function wrapper closure returned by Tymist .tymeth() method.
When .tymth is called it returns associated Tymist .tyme. .tymth provides injected dependency on Tymist tyme base.

`.wind  injects ._tymth dependency from associated Tymist to get its .tyme`

Hidden:

._tymth is injected function wrapper closure returned by .tymen() of
associated Tymist instance that returns Tymist .tyme. when called.

`property tyme`

tyme property getter, get ._tyme .tyme is float cycle time in seconds

`property tymth`

tymth property getter, get ._tymth returns own copy of tymist.tynth function wrapper closure for subsequent injection into related objects that want to be on same tymist tyme base.

`wind`(*tymth*)

Inject new tymist.tymth as new ._tymth. Changes tymist.tyme base. Override in subclasses to update any dependencies on a change in tymist.tymth base

`class hio.base.Tymer`(*duration=None*, *start=None*, ***kwa*)

Bases: *Tymee*

Tymer class to measure cycle time given by .tyme property of Tymist instance. tyme is relative cycle time either artificial or real

Inherited Attributes

Attributes:

Inherited Properties:

.tyme is float relative cycle time of associated Tymist .tyme obtained
via injected .tymth function wrapper closure.

.tymth is function wrapper closure returned by Tymist .tymeth() method.
When .tymth is called it returns associated Tymist .tyme. .tymth provides injected dependency on Tymist tyme base.

Properties:
.duration = tyme duration of tymer in seconds from ._start to ._stop .elaspsed = tyme elasped in seconds since ._start .remaining = tyme remaining in seconds until ._stop .expired = True if expired, False otherwise, i.e. .tyme >= ._stop

Inherited Methods:
.wind is injects ._tymth dependency

`.start() = start tymer at current .tyme`

`.restart() = restart tymer at last ._stop so no time lost`

Hidden:

._tymth is injected function wrapper closure returned by .tymen() of
associated Tymist instance that returns Tymist .tyme. when called.

._start is start tyme in seconds ._stop is stop tyme in seconds

`property duration`

duration property getter, .duration = ._stop - ._start .duration is float duration tyme

property elapsed

elapsed tyme property getter, Returns elapsed tyme in seconds (fractional) since ._start.

property remaining

remaining tyme property getter, Returns remaining tyme in seconds (fractional) before ._stop.

property expired

Returns True if tymer has expired, False otherwise. .tyme >= ._stop,

Duration = 0.0

wind(*tymth*)

Inject new ._tymist and any other bundled tymee references Update any dependencies on a change in ._tymist:

starts over itself at new ._tymists time

start(*duration=None*, *start=None*)

Starts Tymer of duration secs at start time start secs.

If duration not provided then uses current duration If start not provided then starts at current .tyme

restart(*duration=None*)

Lossless restart of Tymer at .tyme = ._stop for duration if provided, current duration otherwise No time lost. Useful to extend Tymer so no time lost

class hio.base.**Doist**(*real=False*, *limit=None*, *doers=None*, ***kwa*)

Bases: *hio.base.tyming.Tymist*

Doist is the root coroutine scheduler Provides relative cycle time in seconds with .tyme property to doers it runs The relative cycle time is advanced in .tock size increments by the by the .tick method. The doist may treat .tyme as artificial time or synchonize it to real time.

.enter method prepares deeds deque of triples (dog, retyme, doer) where

dog is a doer generator returned by calling doer generator instances, functions, or methods.

.recur method runs its deeds deque of triples (dog, retyme, doer) once per

invocation. This synchronizes their cycle time .tyme to the Doist's tyme.

.do method repeatedly runs .recur until generators are complete

it may either repeat as fast as possbile or repeat at real time increments.

Inherited Class Attributes:

.Tock is default .tock

real

True means run in real time, Otherwise as fast as possible.

Type

boolean

limit

maximum run tyme limit then closes all doers

Type

float

done

True means completed due to limit or all deeds completed False is forced complete due to error

Type

boolean

doers

Doer class instances, generator methods or function callables with attributes tock, done, and opts dict(). Used throughout the execution lifecycle.

Type
list

deeds

Tuples of form (dog, retyme, doer). Where: dog is generator created by doer retyme is tyme (real or simulated) in seconds when dog should run next doer is associated doer in .doers list used to assign its .done state

given completion state of its dog

Used throughout the execution lifecycle. The normal case is use the default empty initialization performed here and update in .enter().

Type
deque

timer

for real time intervals

Type
MonoTimer

Inherited Properties:
tyme: is float relative cycle time, .tyme is artificial time : is float tyme increment of .tick()

Properties:

Inherited Methods:
.tick increments .tyme by one .tock or provided tock

.enter prepare deeds, deque of triples (dog, retyme, doer)

.recur run through all deeds once

.do repeadedly call .recur until all dogs in deeds are complete or
times out do to reaching time limit

do(*doers=None*, *limit=None*, *tyme=None*)

Readies deeds deque from .doers or doers if any and then iteratively runs .recur over deeds deque until completion of all deeds. Each entry in deeds is a triple (dog, retyme, doer) where:

dog is generator retyme is tyme (real or simulated) in seconds when dog should run next doer is from .doers list used to assign its .done state given associated completion state of its dog

If interrupted by exception call .close on each dog to force exit context.

Keyboard interrupt (cntl-c) forces exit.

Once finally clause closes a generator it must be reinited before it can be run again

Parameters

- **doers** (*iterable*) – generator method or function callables with attributes tock, done, and opts dict(). This may be used to update the .doers attribute which is used throughout the execution lifecycle. If not provided uses .doers. Parameterization here of doers enables some special cases. The normal case is to initialize in .__init__ or here.
- **limit** (*float*) – is real time limit on execution. Forces close of all dogs.

- **tyme** (*float*) – is optional starting tyme. Resets .tyme to tyme whe provided. If not provided uses current .tyme

Returns
None

See: https://stackoverflow.com/questions/40528867/setting-attributes-on-func For setting attributes on bound methods.

enter(*doers=None*)

Enter context Returns (deque): deeds deque of triples (dog, retyme, doer) where:

dog is generator retyme is tyme (real or simulated) in seconds when dog should run next doer is from .doers list used to assign its .done state given

completion state of its dog

Calls each generator callable (instance or function or method) in .doers to create each generator dog. Injects own tymth function closure, and

generator function's own tock, and opts.

Runs enter context of each dog by calling next(dog)

Parameters

- **attributes** (*doers is list of generator method or function callables with*) – .tock is tyme increment in seconds .done is Boolean completion state .opts is dict() of optional parameters If not provided uses .doers. The normal case is to initialize in .__init__. or .do().
- **triples** (*deeds is deque of deed*) –

Returns
A deed is tuple of form (dog, retyme, doer). If not provided uses .deeds.

Return type
deeds deque()

See: https://stackoverflow.com/questions/40528867/setting-attributes-on-func For setting attributes on bound methods.

recur(*deeds=None*)

Recur once through deeds deque of tuples (triples) of form (dog, retyme, doer) and update in place

Each deed is deque of tuples of form (dog, retyme, doer) where:
dog is generator retyme is tyme (real or simulated) in seconds when dog should run next doer is from .doers list used to assign its .done state given associated completion state of its dog

Each cycle checks all generators in deeds deque and runs if retyme past. At end of cycle advances .tyme by one .tock by calling .tick()

Parameters
deeds (*deque*) – tuples of form (dog, retyme, doer). Parameterization here of deeds enables some special cases.

The Parameterization here of deeds enables some special cases such as manual testing or iteraton. The normal case is to initialize .doers in .__init__. or .do() and to initialize .deeds in .__init__. and then update in .enter()

exit(*deeds=None*)

Force exit each still opened deed calling .close on the dog generator which throws a GeneratorExit to the generator. This executes the close context (GeneratorExit) which then excecutes the exit context in the

finally caluse. Each dogs exit is responsible for releasing resources Previously aborted or closed dogs have already exited Close any running dogs in reverse order so that enters and exits are nested pairs so that the corresponding exits appear in reverse order to their entes. This preserves nested resource dependencies. For example:

enter A,

enter B,
enter C, exit C,

exit B,

exit A

Parameters
deeds (*deque*) – tuples of form (dog, retyme, doer). If not provided uses .deeds. Parameterization here of deeds enables some special cases.

extend(*doers*)

Extend .doers list with doers. Ready deeds from doers and extend .doers and .deeds. Edit deeds in place so not replace deque.

Parameters
extension. (*doers is list of doers to add as*) –

remove(*doers*)

Remove doers from .doers list and any associated deeds from .deeds deque. Force close removed deeds.

Parameters
remove. (*doers is list of doers to*) –

hio.base.**doize**(*, *tock=0.0*, ***opts*)

Returns decorator that makes decorated generator function Doist compatible. Imbues decorated generator function with attributes used by Doist.enter() or DoDoer.enter(). Only one instance of decorated function with shared attributes is allowed.

Usage: @doize def f():

pass

Parameters

- **f** (*tock is default tock attribute of doized*) –
- **attribute** (*opts is dictionary of remaining parameters that becomes .opts*) – of doized f

hio.base.**doify**(*f*, *, *name=None*, *tock=0.0*, ***opts*)

Returns Doist compatible copy, g, of converted generator function f. Each invoction of doify(f) returns a unique copy of doified function f. Imbues copy, g, of converted generator function, f, with attributes used by Doist.enter() or DoDoer.enter(). Allows multiple instances of copy, g, of generator function, f, each with unique attributes.

Usage: def f():

pass

c = doify(f, name='c')

Parameters

- **function** (*f is generator*) –

- **copy** (*name is new function name for returned doified copy g. Default is to*) – f.__name__
- **g** (*tock is default tock attribute of doified copy*) –
- **attribute** (*opts is dictionary of remaining parameters that becomes .opts*) – of doified copy g

Based on: https://stackoverflow.com/questions/972/adding-a-method-to-an-existing-object-instance

class hio.base.**Doer**(*, *tymth=None*, *tock=0.0*, ***opts*)

Bases: *hio.base.tyming.Tymee*

Doer base class for hierarchical structured async coroutine like generators. Doer.__call__ on instance returns generator. Interface for Doist etc is generator function like object. Doer is generator method instance creator and has extra methods and attributes that a plain generator function does not

The .do method executes other methods each corresponding to one of the six econtexts:

enter, recur, clean, exit, (unforced) close, abort (forced)

Actual context order may be one of:
enter, recur, clean, exit enter, recur, close, exit enter, recur, abort, exit enter, abort, exit

.done is Boolean completion state

True means completed Otherwise incomplete. Incompletion maybe due to close or abort.

.opts is dict of injected options into its .do generator by scheduler

Inherited Properties:

.tyme is float relative cycle time of associated Tymist .tyme obtained
via injected .tymth function wrapper closure.

.tymth is function wrapper closure returned by Tymist .tymeth() method.
When .tymth is called it returns associated Tymist .tyme. .tymth provides injected dependency on Tymist tyme base.

Properties:

.tock is float, desired time in seconds between runs or until next run,
non negative, zero means run asap

Inherited Methods:
.wind injects ._tymth dependency from associated Tymist to get its .tyme

.__call__ makes instance callable

Appears as generator function that returns generator

.do is generator method that returns generator

.enter is enter context action method

.recur is recur context action method or generator method

.clean is clean context action method

.exit is exit context method

.close is close context method

`.abort is abort context method`

Hidden:

._tymth is injected function wrapper closure returned by .tymen() of
associated Tymist instance that returns Tymist .tyme. when called.

._tock is hidden attribute for .tock property

`property tock`

tock property getter, get ._tock .tock is float desired .tyme increment in seconds

`__call__`(***kwa*)

Returns generator Does not advance to first yield. The advance to first yield effectively invodes the enter or open context on the generator. To enter either call .next or .send(None) on generator

`do`(*tymth*, *, *tock=0.0*, ***opts*)

Generator method to run this doer. Calling this method returns generator. Interface matches generator function for compatibility. To customize create subclasses and override the lifecycle methods:

.enter, .recur, .exit, .close, .abort

Parameters

- **of** (*tymth is injected function wrapper closure returned by .tymen()*) – Tymist instance. Calling tymth() returns associated Tymist .tyme.
- **value** (*tock is injected initial tock*) –
- **parameters** (*args is dict of injected optional additional*) –

`enter`()

Do 'enter' context actions. Override in subclass. Not a generator method. Set up resources. Comparable to context manager enter.

`recur`(*tyme*)

Do 'recur' context actions. Override in subclass. Regular method that perform repetitive actions once per invocation. Assumes resource setup in .enter() and resource takedown in .exit() (see ReDoer below for example of .recur that is a generator method)

Returns completion state of recurrence actions.
True means done False means continue

Parameters
here. (*Doist feeds its .tyme through .send to .do yield which passes it*) –

.recur maybe implemented by a subclass either as a non-generator method or a generator method. This stub here is as a non-generator method. The base class .do detects which type:

If non-generator .do method runs .recur method once per iteration
until .recur returns (True)

If generator .do method runs .recur with (yield from) until .recur
returns (see ReDoer for example of generator .recur)

clean()

Do 'clean' context actions. Override in subclass. Not a generator method. Clean up resources that are unique to a clean exit. Called by else after normal return.

exit()

Do 'exit' context actions. Override in subclass. Not a generator method. Clean up resources. Comparable to context manager exit. Called by finally after normal return, close, or abort. After .exit() do returns resulting in StopIteration.

close()

Do 'close' context actions. Override in subclass. Not a generator method. Forced close by thrown generator .close() method causing GeneratorExit. .exit() is finally called after .close().

abort(*ex*)

Do 'abort' context actions. Override in subclass. Not a generator method. :param ex is Exception instance that caused abort.:

Unexpected exception that results in generator exiting but not GeneratorExit. .exit() is finally called after .abort().

class hio.base.**DoDoer**(*doers=None*, *always=False*, ***kwa*)

Bases: *Doer*

DoDoer implements Doist like functionality to allow nested scheduling of Doers. Each DoDoer runs a list of doers like a Doist but using the tyme from its

injected tymth for the associated tymist as injected by its ultimate root parent Doist and any intervening parent DoDoer(s).

Scheduling hierarchy: Doist->DoDoer...->DoDoer->Doers

Inherited Attributes:

.done is Boolean completion state:
True means completed Otherwise incomplete. Incompletion maybe due to close or abort.

.opts is dict of injected options for its generator .do

Attributes:

Inherited Properties:

.tyme is float relative cycle time of associated Tymist .tyme obtained
via injected .tymth function wrapper closure.

.tymth is function wrapper closure returned by Tymist .tymeth() method.
When .tymth is called it returns associated Tymist .tyme. .tymth provides injected dependency on Tymist tyme base.

.tock is float, desired time in seconds between runs or until next run,
non negative, zero means run asap

Properties:

doers (list): Doer or Doist compatible generator instances,
functions, or methods.

deeds (deque): tuples of form (dog, retyme, doer) where:
dog is generator created by doer. retyme is tyme in seconds when next should run may be real or simulated. doer is associated doer in .doers list. Used throughout the execution lifecycle. The normal case is use the default empty initialization performed here and update in .enter().

always (bool): True means keep running even when all dogs in deeds
are complete. Enables dynamically managing extending or removing doers and associated deeds while running.

Inherited Methods:
.wind injects ._tymth dependency from associated Tymist to get its .tyme .__call__ makes instance callable

Appears as generator function that returns generator

.do is generator method that returns generator .enter is enter context action method .recur is recur context action method or generator method .clean is clean context action method .exit is exit context method .close is close context method .abort is abort context method

Overidden Methods:
.do .enter .recur .exit

Hidden:

._tymth is injected function wrapper closure returned by .tymen() of
associated Tymist instance that returns Tymist .tyme. when called.

._tock is hidden attribute for .tock property ._always is hidden attribute for .always property ._doers is hidden attribute for .doers property ._deeds is hidden attribute for .deeds property

`property doers`

doers property getter, get ._doers .doers is list of doist compatible generator instances, functions, or

methods

`property deeds`

deeds property getter, get ._deeds .deeds is deque of triples, each of form (dog, retyme, doer)

`property always`

always property getter, get ._always .always is Boolean, True means keep running even when all dogs in deeds

are complete. Enables dynamically managing extending or removing doers and associated deeds while running.

`do`(*tymth*, *tock=0.0*, *doers=None*, *always=None*, ***opts*)

Generator method to run this doer. Equivalent of doist.do Calling this method returns generator Interface matched generator function for compatibility

Parameters

- **of** (*tymth is injected function wrapper closure returned by .tymen()*) – Tymist instance. Calling tymth() returns associated Tymist .tyme.
- **value** (*tock is injected initial tock*) –
- **attributes** (*doers is list of generator method or function callables with*) – tock, done, and opts dict(). This may be used to update the .doers attribute which is used throughout the execution lifecycle. If not provided uses .doers. Parameterization here of doers enables some special cases. The normal case is to initialize in .__init__.
- **deeds** (*always is Boolean. True means keep running even when all dogs in*) – are complete. Enables dynamically managing extending or removing doers and associated deeds while running. When not provided use .always.
- **parameters** (*opts is dict of injected optional additional*) –

enter(*doers=None*)

Do 'enter' context actions. Equivalent of Doist.enter()

Returns deeds deque of triples (dog, retyme, doer) where:
dog is generator created by doer retyme is tyme in seconds when next should run may be real or simulated doer is doer for dog from doers list

Calls each generator callable (function or method) in .doers to create each generator dog.

Runs enter context of each dog by calling next(dog)

Parameters
doers (*list*) – Doer Instance, generator method or function callables with attributes tock, done, and opts dict(). If not provided uses .doers. Parameterization here of doers enables some special cases. The normal case is to initialize in .__init__.

Returns
A deed is tuple of form (dog, retyme, doer). If not provided uses .deeds.

Return type
deeds deque()

See: https://stackoverflow.com/questions/40528867/setting-attributes-on-func For setting attributes on bound methods.

recur(*tyme*, *deeds=None*)

Do 'recur' context actions. Equivalent of Doist.recur

Parameters
tyme (*float*) –

is output of send fed to do yield, The root scheduler
Doist feeds its .tyme which propagates down the chain of DoDoers Because tymist is injected by doist or dodoer, self.tyme is same as tyme. So may use either which is more convenient.

deeds (deque): tuples of form (dog, retyme, doer).
If not provided uses .deeds. Parameterization here of deeds enables some special cases.

Returns completion state of recurrence actions.
True means done False means continue

Cycle once through deeds deque and update in place

Each cycle checks all generators dogs in deeds deque and runs if retyme past.

exit(*deeds=None*)

Do 'exit' context actions.

Parameters
deeds (*deque*) – of deed tuples of form (dog, retyme, doer) If not provided uses .deeds. Parameterization here of deeds enables some special cases.

See: https://stackoverflow.com/questions/40528867/setting-attributes-on-func For setting attributes on bound methods.

extend(*doers*)

Extend .doers list with doers. Ready deeds from doers and extend .doers and .deeds. Edit deeds in place so not replace deque.

Parameters
extension. (*doers is list of doers to add as*) –

remove(*doers*)

Remove doers from .doers list and any associated deeds from .deeds deque. Force close removed deeds.

Parameters

remove. (*doers is list of doers to*) –

hio.base.**openFiler**(*cls=None*, *name='test'*, *temp=True*, *reopen=True*, *clear=False*, ***kwa*)

Context manager wrapper Filer instances for managing a filesystem directory and or files in a directory.

Defaults to using temporary directory path. Context 'with' statements call .close on exit of 'with' block

Parameters

- **instance** (*cls is Class instance of subclass*) –
- **oglers** (*name is str name of ogler instance for filename so can have multiple*) – at different paths thar each use different log file directories
- **Boolean** (*temp is*) – Otherwise open in persistent directory, do not clear on close
- **directory** (*True means open in temporary*) – Otherwise open in persistent directory, do not clear on close
- **close** (*clear on*) – Otherwise open in persistent directory, do not clear on close

Usage:

with openFiler(name="bob") as filer:

with openFiler(name="eve", cls=FilerSubClass) as filer:

class hio.base.**Filer**(*name='main'*, *base=''*, *temp=False*, *headDirPath=None*, *perm=None*, *reopen=True*, *clear=False*, *reuse=False*, *clean=False*, *filed=False*, *mode=None*, *fext=None*, ***kwa*)

Filer instances manage file directories and files to hold keri installation specific resources like databases and configuration files.

name

unique path component used in directory or file path name

Type
str

base

another unique path component inserted before name

Type
str

temp

True means use /tmp directory

Type
bool

headDirPath is head directory path

path is full directory path

perm is numeric os permissions for directory and/or file

Type
s

filed

True means .path ends in file. False means .path ends in directory

Type
bool

mode

file open mode if filed

Type
str

fext

file extension if filed

Type
str

file

Type
File

opened is Boolean, True means directory created and if file then file

is opened. False otherwise

File/Directory Creation Mode Notes:

.Perm provides default restricted access permissions to directory and/or files stat.S_ISVTX | stat.S_IRUSR | stat.S_IWUSR | stat.S_IXUSR 0o1700==960

stat.S_ISVTX is Sticky bit. When this bit is set on a directory it means

that a file in that directory can be renamed or deleted only by the owner of the file, by the owner of the directory, or by a privileged process. When this bit is set on a file it means nothing

stat.S_IRUSR Owner has read permission. stat.S_IWUSR Owner has write permission. stat.S_IXUSR Owner has execute permission.

HeadDirPath = /usr/local/var

TailDirPath = hio

CleanTailDirPath = hio/clean

AltHeadDirPath = ~

AltTailDirPath = .hio

AltCleanTailDirPath = .hio/clean

TempHeadDir = /tmp

TempPrefix = hio_

TempSuffix = _test

Perm

Mode = r+

Fext = text

reopen(*temp=None*, *headDirPath=None*, *perm=None*, *clear=False*, *reuse=False*, *clean=False*, *mode=None*, *fext=None*, ***kwa*)

Open if closed or close and reopen if opened or create and open if not

Parameters

- **temp** (*bool*) – assign to .temp True means open in temporary directory, clear on close False means open persistent directory, do not clear on close
- **headDirPath** (*str*) – optional head directory pathname for main database Default .HeadDirpath
- **perm** (*int*) – optional numeric os dir permissions for database directory and database files. Default .Perm
- **clear** (*bool*) – True means remove directory upon close False means do not remove directory upon close
- **reuse** (*bool*) – True means reuse self.path if already exists False means do not reuse but remake self.path
- **clean** (*bool*) – True means path uses clean tail variant False means path uses normal tail variant
- **mode** (*str*) – file open mode when .filed
- **fext** (*str*) – File extension when .filed

remake(***, *name=''*, *base=''*, *temp=None*, *headDirPath=None*, *perm=None*, *clean=False*, *filed=False*, *mode=None*, *fext=None*, ***kwa*)

Make and return (path. file) by opening or creating and opening if not preexistent, directory or file at path

Parameters

- **name** (*str*) – unique name alias portion of path
- **base** (*str*) – optional base inserted before name in path
- **temp** (*bool*) – optional None means ignore, True means open temporary directory, may clear on close False menans open persistent directory, may not clear on close
- **headDirPath** (*str*) – optional head directory pathname of main database
- **perm** (*int*) – directory or file permissions such as stat.S_IRUSR Owner has read permission. stat.S_IWUSR Owner has write permission. stat.S_IXUSR Owner has execute permission.
- **clean** (*bool*) – True means make path for cleaned version and remove old directory or file at clean path if any. False means make path normally (not clean)
- **filed** (*bool*) – True means .path is file path not directory path False means .path is directiory path not file path
- **mode** (*str*) – file open mode when .filed such as "w+"
- **fext** (*str*) – File extension when .filed

close(*clear=False*)

Close .file if any and if clear rm directory or file at .path

Parameters

clear (*bool*) – True means remove dir or file at .path

_clearPath()

Remove directory/file at end of .path

class hio.base.FilerDoer(*filer*, ***kwa*)

Bases: *hio.base.doing.Doer*

Basic Filer Doer

done

completion state: True means completed Otherwise incomplete. Incompletion maybe due to close or abort.

Type
bool

filer

instance

Type
Filer

Properties:

tyme (float): relative cycle time of associated Tymist .tyme obtained
via injected .tymth function wrapper closure.

tymth (func): closure returned by Tymist .tymeth() method.
When .tymth is called it returns associated Tymist .tyme. .tymth provides injected dependency on Tymist tyme base.

tock (float)): desired time in seconds between runs or until next run,
non negative, zero means run asap

enter()

exit()

hio.core

hio.core Package

Subpackages

hio.core.http

hio.core.http Package

Submodules

hio.core.http.clienting

hio.core.http.clienting module

nonblocking http client

Module Contents

Classes

Requester	Nonblocking HTTP Client Requester class
Respondent	Nonblocking HTTP Client Respondent class
Client	Client class nonblocking HTTP client connection manager and HTTP client
ClientDoer	HTTP Client Doer

Functions

openClient([cls])	Wrapper to create and open HTTP Client instances
backendRequest(tymth, *[, method, scheme, host, port, ...])	Perform Async ReST request to Backend Server

Attributes

logger
CRLF
LF
CR
Response

hio.core.http.clienting.**logger**

hio.core.http.clienting.**CRLF = b'\r\n'**

hio.core.http.clienting.**LF = b'\n'**

hio.core.http.clienting.**CR = b'\r'**

hio.core.http.clienting.**Response**

class hio.core.http.clienting.**Requester**(*hostname='127.0.0.1'*, *port=None*, *scheme='http'*, *method='GET'*, *path='/'*, *qargs=None*, *fragment=''*, *headers=None*, *body=b''*, *data=None*, *fargs=None*, *portOptional=False*)

Bases: object

Nonblocking HTTP Client Requester class

HttpVersionString

Port

reinit(*hostname=None*, *port=None*, *scheme=None*, *method=None*, *path=None*, *qargs=None*, *fragment=None*, *headers=None*, *body=None*, *data=None*, *fargs=None*, *portOptional=None*)

Reinitialize anything that is not None This enables creating another request on a connection to the same host port

rebuild(*method=None*, *path=None*, *qargs=None*, *fragment=None*, *headers=None*, *body=None*, *data=None*, *fargs=None*, *portOptional=None*)

Reinit then build and return request message This allows sending another request to same destination

build()

Build and return request message from attributes

class hio.core.http.clienting.**Respondent**(*redirects=None*, *redirectable=True*, *events=None*, *retry=None*, *leid=None*, ***kwa*)

Bases: *hio.core.http.httping.Parsent*

Nonblocking HTTP Client Respondent class

Retry = 100

reinit(*redirectable=None*, ***kwa*)

Reinitialize Instance See super class redirectable means allow redirection

close()

Call super to assign True to .closed Also close event source

checkPersisted()

Checks headers to determine if connection should be kept open until server closes it Sets the .persisted flag

parseHead()

Generator to parse headers in heading of .msg Yields None if more to parse Yields True if done parsing

parseBody()

Parse body

hio.core.http.clienting.**openClient**(*cls=None*, ***kwa*)

Wrapper to create and open HTTP Client instances When used in with statement block, calls .close() on exit of with block

Parameters

instance (*cls is Class instance of subclass*) –

Usage:

with openClient() as client0:
client0.accept()

with openClient(cls=Client) as client0:
client0.accept()

class hio.core.http.clienting.**Client**(*connector=None*, *requester=None*, *respondent=None*, *name=''*, *uid=0*, *bufsize=8096*, *wl=None*, *hostname='127.0.0.1'*, *port=None*, *scheme=''*, *method='GET'*, *path='/'*, *headers=None*, *qargs=None*, *fragment=''*, *body=b''*, *data=None*, *fargs=None*, *requests=None*, *msg=None*, *dictable=None*, *events=None*, *redirectable=True*, *redirects=None*, *responses=None*, *portOptional=False*, ***kwa*)

Client class nonblocking HTTP client connection manager and HTTP client request and response processor

wind(*tymth*)

Inject new tymist.tymth as new ._tymth. Changes tymist.tyme base. Updates winds .tymer .tymth

reopen()

Return result of .connector.reopen()

close()

Call .connecter.close (shutclose)

static attrify(*response*)

Convert response dict into named tuple Response so can access fields as attributes

respond()

Pops and returns next response from .responses if any Otherwise returns None

request(*method=None*, *path=None*, *qargs=None*, *fragment=None*, *headers=None*, *body=None*, *data=None*, *fargs=None*, *reply=None*, ***kwa*)

Create and append request dict onto .requests with updated values from parameters. Use existing .requester values if not provided except for body. Body/Data/fargs must be newly provided. This is a differential request that reuses latest values if not changed. Requires that patron already be setup with scheme host port

request = dict([('method', method),
('path', path), ('qargs', dict([("auth", self.token.value)])), ('headers', dict([('Accept', 'application/json'),

('Connection', 'Keep-Alive'), ('Keep-Alive', 'timeout=60, max=100'),])),

('body', body), ('reply', dict([('rid', rid), ('rdeck', replies)])),])

self.patron.value.requests.append(request)

transmit(*method=None*, *path=None*, *qargs=None*, *fragment=None*, *headers=None*, *body=None*, *data=None*, *fargs=None*, ***kwa*)

Rebuild and transmit request Assumes that .waited is not True. Do not use bare unless know that there is no current request/reponse in process otherwise it clobbers .requester attributes

If the parameters are all None then use existing .requester and .respondent attributes otherwise reinit .requester and .respondent if method is not None

Should only use with all None first time after that change one of the parameters.

redirect()

Perform redirect

serviceRequests()

Service requests deque

serviceResponse()

Service Rx on connection and parse

service()

Service request response

serviceWhileGen(*tymeout=0.5*)

Generator Method. ServiceAll while pending requests or not a response or not tymeout

Usage: response = yield from .serviceWhileGen(tymeout=0.5)

Runs one iteration of .service() on next and yields empty bytes while not done.

Assumes associated tymist is advanced and realtime sleep delay is incurred elsewhere.

Returns response as namedtuple or None if tymeout.

hio.core.http.clienting.**backendRequest**(*tymth*, ***, *method='GET'*, *scheme=''*, *host='localhost'*, *port=None*, *path='/'*, *qargs=None*, *data=None*, *buffered=False*, *tymeout=2.0*)

Perform Async ReST request to Backend Server

Parameters:

Usage: (Inside a generator function)

response = yield from backendRequest()

response is the response if valid else None before response is completed the yield from yields up an empty string '' once completed then response has a value

path can be full url with host port etc path takes precedence over others

class hio.core.http.clienting.**ClientDoer**(*client*, ***kwa*)

Bases: *hio.base.doing.Doer*

HTTP Client Doer

See Doer for inherited attributes, properties, and methods.

.client is HTTP Client instance

wind(*tymth*)

Inject new tymist.tymth as new ._tymth. Changes tymist.tyme base. Updates winds .tymer .tymth

enter()

recur(*tyme*)

exit()

hio.core.http.httping

hio.core.http.httping module http async io (nonblocking) support

Module Contents

Classes

EventSource	Server Sent Event Stream Client parser
Parsent	Base class for objects that parse HTTP messages

Functions

httpDate1123(dt)	Return a string representation of a date according to RFC 1123
normalizeHostPort(host[, port, defaultPort])	Given hostname host which could also be netloc which includes port
parseQuery(query)	Return dict of parsed query string.
updateQargsQuery([qargs, query])	Returns duple of updated (qargs, query)
unquoteQuery(query)	Returns query string with unquoted values
packHeader(name, *values)	Format and return a header line.
packChunk(msg)	Return msg bytes in a chunk
parseLine(raw[, eols, kind])	Generator to parse line from raw bytearray
parseLeader(raw[, eols, kind, headers])	Generator to parse entire leader of header lines from raw bytearray
parseChunk(raw)	Generator to parse next chunk from raw bytearray
parseBom(raw[, bom])	Generator to parse bom from raw bytearray
parseStatusLine(line)	Parse the response status line
parseRequestLine(line)	Parse the request start line

Attributes

CRLF
LF
CR
MAX_LINE_SIZE
MAX_HEADERS
HTTP_PORT
HTTPS_PORT
HTTP_11_VERSION_STRING
CONTINUE

continues on next page

Table 1 – continued from previous page

SWITCHING_PROTOCOLS
PROCESSING
OK
CREATED
ACCEPTED
NON_AUTHORITATIVE_INFORMATION
NO_CONTENT
RESET_CONTENT
PARTIAL_CONTENT
MULTI_STATUS
IM_USED
MULTIPLE_CHOICES
MOVED_PERMANENTLY
FOUND
SEE_OTHER
NOT_MODIFIED
USE_PROXY
TEMPORARY_REDIRECT
BAD_REQUEST
UNAUTHORIZED
PAYMENT_REQUIRED
FORBIDDEN
NOT_FOUND
METHOD_NOT_ALLOWED
NOT_ACCEPTABLE

continues on next page

Table 1 – continued from previous page

PROXY_AUTHENTICATION_REQUIRED
REQUEST_TIMEOUT
CONFLICT
GONE
LENGTH_REQUIRED
PRECONDITION_FAILED
REQUEST_ENTITY_TOO_LARGE
REQUEST_URI_TOO_LONG
UNSUPPORTED_MEDIA_TYPE
REQUESTED_RANGE_NOT_SATISFIABLE
EXPECTATION_FAILED
UNPROCESSABLE_ENTITY
LOCKED
FAILED_DEPENDENCY
UPGRADE_REQUIRED
PRECONDITION_REQUIRED
TOO_MANY_REQUESTS
REQUEST_HEADER_FIELDS_TOO_LARGE
INTERNAL_SERVER_ERROR
NOT_IMPLEMENTED
BAD_GATEWAY
SERVICE_UNAVAILABLE
GATEWAY_TIMEOUT
HTTP_VERSION_NOT_SUPPORTED
INSUFFICIENT_STORAGE

continues on next page

Table 1 – continued from previous page

NOT_EXTENDED
NETWORK_AUTHENTICATION_REQUIRED
STATUS_DESCRIPTIONS
METHODS
MAXAMOUNT
_MAXLINE
_MAXHEADERS

hio.core.http.httping.**CRLF = b'\r\n'**

hio.core.http.httping.**LF = b'\n'**

hio.core.http.httping.**CR = b'\r'**

hio.core.http.httping.**MAX_LINE_SIZE = 65536**

hio.core.http.httping.**MAX_HEADERS = 100**

hio.core.http.httping.**HTTP_PORT = 80**

hio.core.http.httping.**HTTPS_PORT = 443**

hio.core.http.httping.**HTTP_11_VERSION_STRING = HTTP/1.1**

hio.core.http.httping.**CONTINUE = 100**

hio.core.http.httping.**SWITCHING_PROTOCOLS = 101**

hio.core.http.httping.**PROCESSING = 102**

hio.core.http.httping.**OK = 200**

hio.core.http.httping.**CREATED = 201**

hio.core.http.httping.**ACCEPTED = 202**

hio.core.http.httping.**NON_AUTHORITATIVE_INFORMATION = 203**

hio.core.http.httping.**NO_CONTENT = 204**

hio.core.http.httping.**RESET_CONTENT = 205**

hio.core.http.httping.**PARTIAL_CONTENT = 206**

hio.core.http.httping.**MULTI_STATUS = 207**

hio.core.http.httping.**IM_USED = 226**

hio.core.http.httping.**MULTIPLE_CHOICES = 300**

hio.core.http.httping.**MOVED_PERMANENTLY = 301**

hio.core.http.httping.**FOUND = 302**

hio.core.http.httping.**SEE_OTHER = 303**

hio.core.http.httping.**NOT_MODIFIED = 304**

hio.core.http.httping.**USE_PROXY = 305**

hio.core.http.httping.**TEMPORARY_REDIRECT = 307**

hio.core.http.httping.**BAD_REQUEST = 400**

hio.core.http.httping.**UNAUTHORIZED = 401**

hio.core.http.httping.**PAYMENT_REQUIRED = 402**

hio.core.http.httping.**FORBIDDEN = 403**

hio.core.http.httping.**NOT_FOUND = 404**

hio.core.http.httping.**METHOD_NOT_ALLOWED = 405**

hio.core.http.httping.**NOT_ACCEPTABLE = 406**

hio.core.http.httping.**PROXY_AUTHENTICATION_REQUIRED = 407**

hio.core.http.httping.**REQUEST_TIMEOUT = 408**

hio.core.http.httping.**CONFLICT = 409**

hio.core.http.httping.**GONE = 410**

hio.core.http.httping.**LENGTH_REQUIRED = 411**

hio.core.http.httping.**PRECONDITION_FAILED = 412**

hio.core.http.httping.**REQUEST_ENTITY_TOO_LARGE = 413**

hio.core.http.httping.**REQUEST_URI_TOO_LONG = 414**

hio.core.http.httping.**UNSUPPORTED_MEDIA_TYPE = 415**

hio.core.http.httping.**REQUESTED_RANGE_NOT_SATISFIABLE = 416**

hio.core.http.httping.**EXPECTATION_FAILED = 417**

hio.core.http.httping.**UNPROCESSABLE_ENTITY = 422**

hio.core.http.httping.**LOCKED = 423**

hio.core.http.httping.**FAILED_DEPENDENCY = 424**

hio.core.http.httping.**UPGRADE_REQUIRED = 426**

hio.core.http.httping.**PRECONDITION_REQUIRED = 428**

hio.core.http.httping.**TOO_MANY_REQUESTS = 429**

hio.core.http.httping.**REQUEST_HEADER_FIELDS_TOO_LARGE = 431**

hio.core.http.httping.**INTERNAL_SERVER_ERROR = 500**

hio.core.http.httping.**NOT_IMPLEMENTED = 501**

hio.core.http.httping.**BAD_GATEWAY = 502**

hio.core.http.httping.**SERVICE_UNAVAILABLE = 503**

hio.core.http.httping.**GATEWAY_TIMEOUT = 504**

hio.core.http.httping.**HTTP_VERSION_NOT_SUPPORTED = 505**

hio.core.http.httping.**INSUFFICIENT_STORAGE = 507**

hio.core.http.httping.**NOT_EXTENDED = 510**

hio.core.http.httping.**NETWORK_AUTHENTICATION_REQUIRED = 511**

hio.core.http.httping.**STATUS_DESCRIPTIONS**

hio.core.http.httping.**METHODS = ['GET', 'HEAD', 'PUT', 'PATCH', 'POST', 'DELETE', 'OPTIONS', 'TRACE', 'CONNECT']**

hio.core.http.httping.**MAXAMOUNT = 1048576**

hio.core.http.httping.**_MAXLINE = 65536**

hio.core.http.httping.**_MAXHEADERS = 100**

exception hio.core.http.httping.**HTTPException**

Bases: Exception

Common base class for all non-exit exceptions.

exception hio.core.http.httping.**InvalidURL**

Bases: *HTTPException*

Common base class for all non-exit exceptions.

exception hio.core.http.httping.**UnknownProtocol**(*version*)

Bases: *HTTPException*

Common base class for all non-exit exceptions.

exception hio.core.http.httping.**BadStatusLine**(*line*)

Bases: *HTTPException*

Common base class for all non-exit exceptions.

exception hio.core.http.httping.**BadRequestLine**(*line*)

Bases: *BadStatusLine*

Common base class for all non-exit exceptions.

exception hio.core.http.httping.**BadMethod**(*method*)

Bases: *HTTPException*

Common base class for all non-exit exceptions.

exception hio.core.http.httping.**LineTooLong**(*kind*)

Bases: *HTTPException*

Common base class for all non-exit exceptions.

exception hio.core.http.httping.**PrematureClosure**(*msg*)

Bases: *HTTPException*

Common base class for all non-exit exceptions.

exception hio.core.http.httping.**HTTPError**(*status*, *reason=''*, *title=''*, *detail=''*, *fault=None*, *headers=None*)

Bases: Exception

HTTP error for use with Valet or Other WSGI servers to raise exceptions caught by the WSGI server.

status is int HTTP status code, e.g. 400

reason is str HTTP status text, "Unknown Error"

title is str title of error

headers is dict of extra headers to add to the response

error

An internal application error code

Type
int

__slots__ = ['status', 'reason', 'title', 'detail', 'headers', 'fault']

__repr__()

Return repr(self).

render(*jsonify=False*)

Render and return the attributes as a bytes If jsonify then render as serialized json

hio.core.http.httping.**httpDate1123**(*dt*)

Return a string representation of a date according to RFC 1123 (HTTP/1.1).

The supplied date must be in UTC. import datetime httpDate1123(datetime.datetime.utcnow()) 'Wed, 30 Sep 2015 14:29:18 GMT'

hio.core.http.httping.**normalizeHostPort**(*host*, *port=None*, *defaultPort=80*)

Given hostname host which could also be netloc which includes port and or port generate and return tuple (hostname, port) priority is if port is provided in hostname as host:port then use otherwise use port otherwise use defaultPort

hio.core.http.httping.**parseQuery**(*query*)

Return dict of parsed query string. Utility function

hio.core.http.httping.**updateQargsQuery**(*qargs=None*, *query=''*)

Returns duple of updated (qargs, query) Where qargs parameter is dict of query arguments and query parameter is query string The returned qargs is updated with query string arguments and the returned query string is generated from the updated qargs If provided, qargs may have additional fields not in query string This allows combining query args from two sources, a dict and a string

https://www.w3.org/TR/2014/REC-html5-20141028/forms.html#url-encoded-form-data

hio.core.http.httping.**unquoteQuery**(*query*)

Returns query string with unquoted values

hio.core.http.httping.**packHeader**(*name*, **values*)

Format and return a header line.

For example: h.packHeader('Accept', 'text/html')

hio.core.http.httping.**packChunk**(*msg*)

Return msg bytes in a chunk

hio.core.http.httping.**parseLine**(*raw*, *eols=(CRLF, LF, CR)*, *kind='event line'*)

Generator to parse line from raw bytearray Each line demarcated by one of eols kind is line type string for error message

Yields None If waiting for more to parse Yields line Otherwise

Consumes parsed portions of raw bytearray

Raise error if eol not found before MAX_LINE_SIZE

hio.core.http.httping.**parseLeader**(*raw*, *eols=(CRLF, LF)*, *kind='leader header line'*, *headers=None*)

Generator to parse entire leader of header lines from raw bytearray Each line demarcated by one of eols Yields None If more to parse Yields cimdict of headers Otherwise as indicated by empty headers

Raise error if eol not found before MAX_LINE_SIZE

hio.core.http.httping.**parseChunk**(*raw*)

Generator to parse next chunk from raw bytearray Consumes used portions of raw Yields None If waiting for more bytes Yields tuple (size, parms, trails, chunk) Otherwise Where:

size is int size of the chunk parms is dict of chunk extension parameters trails is dict of chunk trailer headers (only on last chunk if any) chunk is chunk if any or empty if not

Chunked-Body = *chunk
last-chunk trailer CRLF

chunk = chunk-size [chunk-extension] CRLF
chunk-data CRLF

chunk-size = 1*HEX last-chunk = 1*("0") [chunk-extension] CRLF chunk-extension= *(";" chunk-ext-name ["=" chunk-ext-val]) chunk-ext-name = token chunk-ext-val = token | quoted-string chunk-data = chunk-size(OCTET) trailer = *(entity-header CRLF)

hio.core.http.httping.**parseBom**(*raw*, *bom=codecs.BOM_UTF8*)

Generator to parse bom from raw bytearray Yields None If waiting for more to parse Yields bom If found Yields empty bytearray Otherwise Consumes parsed portions of raw bytearray

hio.core.http.httping.**parseStatusLine**(*line*)

Parse the response status line

hio.core.http.httping.**parseRequestLine**(*line*)

Parse the request start line

class hio.core.http.httping.**EventSource**(*raw=None*, *events=None*, *dictable=False*)

Bases: `object`

Server Sent Event Stream Client parser

Bom

`close()`

Assign True to .closed

`parseEvents()`

Generator to parse events from .raw bytearray and append to .events Each event is dict with the following items:

id: event id utf-8 decoded or empty

name: event name utf-8 decoded or empty data: event data utf-8 decoded json: event data deserialized to dict when applicable pr None

assigns .retry if any

Yields None If waiting for more bytes Yields True When done

event = *(comment / field) end-of-line comment = colon *any-char end-of-line field = 1*name-char [colon [space] *any-char] end-of-line end-of-line = (cr lf / cr / lf / eof) eof = < matches repeatedly at the end of the stream > lf =

0xA

cr =

0xD

space = 0x20 colon = 0x3A bom = when encoded as utf-8 b'ï»¿' name-char = a Unicode character other than LF, CR, or : any-char = a Unicode character other than LF or CR Event streams in this format must always be encoded as UTF-8. [RFC3629]

`parseEventStream()`

Generator to parse event stream from .raw bytearray stream appends each event to .events deque. assigns .bom if any assigns .retry if any Parses until connection closed

Each event is dict with the following items:

id: event id utf-8 decoded or empty

name: event name utf-8 decoded or empty data: event data utf-8 decoded json: event data deserialized to dict when applicable pr None

Yields None If waiting for more bytes Yields True When completed and sets .ended to True If BOM present at beginning of event stream then assigns to .bom and deletes. Consumes bytearray as it parses

stream = [bom] *event event = *(comment / field) end-of-line comment = colon *any-char end-of-line field = 1*name-char [colon [space] *any-char] end-of-line end-of-line = (cr lf / cr / lf / eof) eof = < matches repeatedly at the end of the stream > lf =

0xA

cr =

0xD

space = 0x20 colon = 0x3A bom = when encoded as utf-8 b'ï»¿' name-char = a Unicode character other than LF, CR, or : any-char = a Unicode character other than LF or CR Event streams in this format must always be encoded as UTF-8. [RFC3629]

`makeParser`(*raw=None*)

Make event stream parser generator and assign to .parser Assign msg to .msg If provided

parse()

Service the event stream parsing must call .makeParser to setup parser When done parsing,

.parser is None .ended is True

class hio.core.http.httping.**Parsent**(*msg=None*, *dictable=None*, *method='GET'*)

Bases: object

Base class for objects that parse HTTP messages

reinit(*msg=None*, *dictable=None*, *method='GET'*)

Reinitialize Instance msg = bytearray of request msg to parse dictable = Boolean flag If True attempt to convert json body method = method verb of associated request

close()

Assign True to .closed and close parser

checkPersisted()

Checks headers to determine if connection should be kept open until client closes it Sets the .persisted flag

parseHead()

Generator to parse headers in heading of .msg Yields None if more to parse Yields True if done parsing

parseBody()

Parse body

parseMessage()

Generator to parse message bytearray. Parses msg if not None Otherwise parse .msg

makeParser(*msg=None*)

Make message parser generator and assign to .parser Assign msg to .msg If provided

parse()

Service the message parsing must call .makeParser to setup parser When done parsing,

.parser is None .ended is True

dictify()

Attempt to convert body to dict data if .dictable or json content-type

hio.core.http.serving

hio.core.http.serving classes

nonblocking http server

Module Contents

Classes

Requestant	Nonblocking HTTP Server Requestant class
Responder	Nonblocking HTTP WSGI Responder class
Server	Server WSGI HTTP Server Class
CustomResponder	Nonblocking HTTP Server Response class for non-wsgi applications
Steward	Manages the associated requestant and responder for an incoming connection
BareServer	BareServer class nonblocking Bare (non-WSGI) HTTP server
StaticSink	Class that provides Falcon sink endpoint for serving static files in support
ServerDoer	HTTP WSGI Server Doer

Functions

openServer([cls])	Wrapper to create and open HTTP Server instances

Attributes

logger
CRLF
LF
CR
WsgiServer

hio.core.http.serving.**logger**

hio.core.http.serving.**CRLF = b'\r\n'**

hio.core.http.serving.**LF = b'\n'**

hio.core.http.serving.**CR = b'\r'**

class hio.core.http.serving.**Requestant**(*remoter=None*, ***kwa*)

Bases: *hio.core.http.httping.Parsent*

Nonblocking HTTP Server Requestant class Parses request msg

checkPersisted()

Checks headers to determine if connection should be kept open until client closes it Sets the .persisted flag

parseHead()

Generator to parse headers in heading of .msg Yields None if more to parse Yields True if done parsing

parseBody()

Parse body

class hio.core.http.serving.**Responder**(*incomer*, *app*, *environ*, *chunkable=False*, *delay=None*)

Nonblocking HTTP WSGI Responder class

HttpVersionString

Delay = 1.0

close()

Close any resources

reset(*environ*, *chunkable=None*)

Reset attributes for another request-response

build()

Return built head bytes from .status and .headers

write(*msg*)

WSGI write callback This writes out the headers the first time its called otherwise writes the msg bytes

start(*status*, *response_headers*, *exc_info=None*)

WSGI application start_response callable

Parameters:

status is string of status code and status reason '200 OK' or simple
status code int which will be replaced with string

response_headers is list of tuples of strings of the form (field, value)
one tuple for each header example: [

('Content-type', 'text/plain'), ('X-Some-Header', 'value')

]

exc_info is optional exception info if exception occurred while
processing request in wsgi application If exc_info is supplied, and no HTTP headers have been output yet, start_response should replace the currently-stored HTTP response headers with the newly-supplied ones, thus allowing the application to "change its mind" about the output when an error has occurred.

However, if exc_info is provided, and the HTTP headers have already been sent, start_response must raise an error, and should re-raise using the exc_info tuple. That is:

raise exc_info[1].with_traceback(exc_info[2]) (python3)

Nonstandard modifiction to allow for iterable/generator of body to change
headers and status before first write to support async processing of responses whose iterator/generator yields empty before first non-empty yield. In .service yielding empty does not cause write so status line and headers are not sent until first non-empty write.

The mode is that the app.headers and app.status are consulted to see if changed from when .start = wsgi start_response was first called.

service()

Service wsgi compatible application

hio.core.http.serving.**openServer**(*cls=None*, ***kwa*)

Wrapper to create and open HTTP Server instances When used in with statement block, calls .close() on exit of with block

Parameters
instance (*cls is Class instance of subclass*) –

Usage:

with openServer() as server0:
server0.

with openServer(cls=BareServer) as server0:
server0.

class hio.core.http.serving.**Server**(*name='hio.wsgi.server'*, *app=None*, *reqs=None*, *reps=None*, *servant=None*, *bufsize=8096*, *wl=None*, *ha=None*, *host=''*, *port=None*, *eha=None*, *scheme=''*, *tymeout=None*, ***kwa*)

Server WSGI HTTP Server Class

Tymeout = 5.0

wind(*tymth*)
Inject new tymist.tymth as new ._tymth. Changes tymist.tyme base. Updates winds .tymer .tymth

reopen()
Return result of .servant.reopen()

close()
Close all reqs, reps, and ixes

idle()
Returns True if no connections have requests in process Useful for debugging

buildEnviron(*requestant*)
Returns wisgi environment dictionary for supplied requestant

closeConnection(*ca*)
Close and remove connection given by ca

serviceConnects()
Service new incoming connections Create requestants Timeout stale connections

serviceReqs()
Service pending requestants

serviceReps()
Service pending responders

service()
Service request response

hio.core.http.serving.**WsgiServer**

class hio.core.http.serving.**CustomResponder**(*steward=None*, *status=200*, *headers=None*, *body=b''*, *data=None*)

Nonblocking HTTP Server Response class for non-wsgi applications Used by Steward

HTTP/1.1 200 OK

Content-Length: 122

Content-Type: application/json

Date: Thu, 30 Apr 2015 19:37:17 GMT

Server: IoBook.local

HttpVersionString

reinit(*status=None*, *headers=None*, *body=None*, *data=None*)

Reinitialize anything that is not None This enables creating another response on a connection

build(*status=None*, *headers=None*, *body=None*, *data=None*)

Build and return response message

class hio.core.http.serving.**Steward**(*remoter*, *requestant=None*, *responder=None*, *dictable=False*)

Manages the associated requestant and responder for an incoming connection for BareServer (non-wsgi) HTTP server

refresh()

Restart incomer tymer

respond()

Respond to request Override in subclass Echo request

pour()

Run generator to stream response message

class hio.core.http.serving.**BareServer**(*servant=None*, *stewards=None*, *name=''*, *bufsize=8096*, *wl=None*, *ha=None*, *host=''*, *port=None*, *eha=None*, *scheme=''*, *dictable=False*, *timeout=None*, ***kwa*)

BareServer class nonblocking Bare (non-WSGI) HTTP server Define CustomResponder subclass to respond to requests as per Steward

Timeout = 5.0

reopen()

Return result of .servant.reopen()

idle()

Returns True if no connections have requests in process Useful for debugging

close()

Close all ixes for all stewards

closeConnection(*ca*)

Close and remove connection and associated steward given by ca

serviceConnects()

Service new incoming connections Create requestants Timeout stale connections

serviceStewards()

Service pending requestants and responders

service()

Service request response

class hio.core.http.serving.**StaticSink**(*staticDirPath=None*)

Class that provides Falcon sink endpoint for serving static files in support of a client side web app.

StaticSinkBasePath = /static

DefaultStaticSinkBasePath = /

__call__(*req*, *rep*)

class hio.core.http.serving.**ServerDoer**(*server*, ***kwa*)

Bases: *hio.base.doing.Doer*

HTTP WSGI Server Doer

See Doer for inherited attributes, properties, and methods.

.server is HTTP WSGI Server instance

Properties:

wind(*tymth*)

Inject new tymist.tymth as new ._tymth. Changes tymist.tyme base. Updates winds .tymer .tymth

enter()

recur(*tyme*)

exit()

Package Contents

Classes

Client	Client class nonblocking HTTP client connection manager and HTTP client
ClientDoer	HTTP Client Doer
BareServer	BareServer class nonblocking Bare (non-WSGI) HTTP server
Server	Server WSGI HTTP Server Class
ServerDoer	HTTP WSGI Server Doer

Functions

openClient([cls])	Wrapper to create and open HTTP Client instances
openServer([cls])	Wrapper to create and open HTTP Server instances

Attributes

WsgiServer

exception hio.core.http.**HTTPError**(*status*, *reason=''*, *title=''*, *detail=''*, *fault=None*, *headers=None*)

Bases: Exception

HTTP error for use with Valet or Other WSGI servers to raise exceptions caught by the WSGI server.

status is int HTTP status code, e.g. 400

reason is str HTTP status text, "Unknown Error"

title is str title of error

headers is dict of extra headers to add to the response

error

An internal application error code

Type
int

__slots__ = ['status', 'reason', 'title', 'detail', 'headers', 'fault']

__repr__()

Return repr(self).

render(*jsonify=False*)

Render and return the attributes as a bytes If jsonify then render as serialized json

class hio.core.http.**Client**(*connector=None*, *requester=None*, *respondent=None*, *name=''*, *uid=0*, *bufsize=8096*, *wl=None*, *hostname='127.0.0.1'*, *port=None*, *scheme=''*, *method='GET'*, *path='/'*, *headers=None*, *qargs=None*, *fragment=''*, *body=b''*, *data=None*, *fargs=None*, *requests=None*, *msg=None*, *dictable=None*, *events=None*, *redirectable=True*, *redirects=None*, *responses=None*, *portOptional=False*, ***kwa*)

Client class nonblocking HTTP client connection manager and HTTP client request and response processor

wind(*tymth*)

Inject new tymist.tymth as new ._tymth. Changes tymist.tyme base. Updates winds .tymer .tymth

reopen()

Return result of .connector.reopen()

close()

Call .connecter.close (shutclose)

static attrify(*response*)

Convert response dict into named tuple Response so can access fields as attributes

respond()

Pops and returns next response from .responses if any Otherwise returns None

request(*method=None*, *path=None*, *qargs=None*, *fragment=None*, *headers=None*, *body=None*, *data=None*, *fargs=None*, *reply=None*, ***kwa*)

Create and append request dict onto .requests with updated values from parameters. Use existing .requester values if not provided except for body. Body/Data/fargs must be newly provided. This is a differential request that reuses latest values if not changed. Requires that patron already be setup with scheme host port

request = dict([('method', method),
('path', path), ('qargs', dict([("auth", self.token.value)])), ('headers', dict([('Accept', 'application/json'),

('Connection', 'Keep-Alive'), ('Keep-Alive', 'timeout=60, max=100'),])),

('body', body), ('reply', dict([('rid', rid), ('rdeck', replies)])),])

self.patron.value.requests.append(request)

transmit(*method=None*, *path=None*, *qargs=None*, *fragment=None*, *headers=None*, *body=None*, *data=None*, *fargs=None*, ***kwa*)

Rebuild and transmit request Assumes that .waited is not True. Do not use bare unless know that there is no current request/reponse in process otherwise it clobbers .requester attributes

If the parameters are all None then use existing .requester and .respondent attributes otherwise reinit .requester and .respondent if method is not None

Should only use with all None first time after that change one of the parameters.

redirect()

Perform redirect

serviceRequests()

Service requests deque

serviceResponse()

Service Rx on connection and parse

service()

Service request response

serviceWhileGen(*tymeout=0.5*)

Generator Method. ServiceAll while pending requests or not a response or not tymeout

Usage: response = yield from .serviceWhileGen(tymeout=0.5)

Runs one iteration of .service() on next and yields empty bytes while not done.

Assumes associated tymist is advanced and realtime sleep delay is incurred elsewhere.

Returns response as namedtuple or None if tymeout.

hio.core.http.**openClient**(*cls=None*, ***kwa*)

Wrapper to create and open HTTP Client instances When used in with statement block, calls .close() on exit of with block

Parameters
instance (*cls is Class instance of subclass*) –

Usage:

with openClient() as client0:
client0.accept()

with openClient(cls=Client) as client0:
client0.accept()

class hio.core.http.**ClientDoer**(*client*, ***kwa*)

Bases: *hio.base.doing.Doer*

HTTP Client Doer

See Doer for inherited attributes, properties, and methods.

.client is HTTP Client instance

wind(*tymth*)

Inject new tymist.tymth as new ._tymth. Changes tymist.tyme base. Updates winds .tymer .tymth

enter()

recur(*tyme*)

exit()

class hio.core.http.**BareServer**(*servant=None*, *stewards=None*, *name=''*, *bufsize=8096*, *wl=None*, *ha=None*, *host=''*, *port=None*, *eha=None*, *scheme=''*, *dictable=False*, *timeout=None*, ***kwa*)

BareServer class nonblocking Bare (non-WSGI) HTTP server Define CustomResponder subclass to respond to requests as per Steward

Timeout = 5.0

reopen()

Return result of .servant.reopen()

idle()

Returns True if no connections have requests in process Useful for debugging

close()

Close all ixes for all stewards

closeConnection(*ca*)

Close and remove connection and associated steward given by ca

serviceConnects()

Service new incoming connections Create requestants Timeout stale connections

serviceStewards()

Service pending requestants and responders

service()

Service request response

class hio.core.http.**Server**(*name='hio.wsgi.server'*, *app=None*, *reqs=None*, *reps=None*, *servant=None*, *bufsize=8096*, *wl=None*, *ha=None*, *host=''*, *port=None*, *eha=None*, *scheme=''*, *tymeout=None*, ***kwa*)

Server WSGI HTTP Server Class

Tymeout = 5.0

wind(*tymth*)

Inject new tymist.tymth as new ._tymth. Changes tymist.tyme base. Updates winds .tymer .tymth

reopen()

Return result of .servant.reopen()

close()

Close all reqs, reps, and ixes

idle()

Returns True if no connections have requests in process Useful for debugging

buildEnviron(*requestant*)

Returns wisgi environment dictionary for supplied requestant

closeConnection(*ca*)

Close and remove connection given by ca

serviceConnects()

Service new incoming connections Create requestants Timeout stale connections

serviceReqs()

Service pending requestants

serviceReps()

Service pending responders

service()

Service request response

hio.core.http.**WsgiServer**

hio.core.http.**openServer**(*cls=None*, ***kwa*)

Wrapper to create and open HTTP Server instances When used in with statement block, calls .close() on exit of with block

Parameters

instance (*cls is Class instance of subclass*) –

Usage:

with openServer() as server0:
server0.

with openServer(cls=BareServer) as server0:
server0.

class hio.core.http.**ServerDoer**(*server*, ***kwa*)

Bases: *hio.base.doing.Doer*

HTTP WSGI Server Doer

See Doer for inherited attributes, properties, and methods.

.server is HTTP WSGI Server instance

Properties:

wind(*tymth*)

Inject new tymist.tymth as new ._tymth. Changes tymist.tyme base. Updates winds .tymer .tymth

enter()

recur(*tyme*)

exit()

hio.core.serial

hio.core.serial Package

Submodules

hio.core.serial.serialing

Asynchronous (nonblocking) serial io

Module Contents

Classes

Console	Class to manage non blocking interactive I/O on serial console
ConsoleDoer	Basic Console Doer. Wraps console in doer context so opens and closes console
EchoConsoleDoer	Basic Terminal Console IO to buffers. Echos input back to output
Device	Class to manage non blocking IO on serial device port.
Serial	Class to manage non blocking IO on serial device port using pyserial
Driver	Nonblocking Serial Device Port Driver

Attributes

logger	

hio.core.serial.serialing.**logger**

exception hio.core.serial.serialing.**LineError**

Bases: *hio.hioing.HioError*

Serial line error. Too big for buffer.

Usage:
raise LineError("error message")

class hio.core.serial.serialing.**Console**(*bs=None*)

Class to manage non blocking interactive I/O on serial console

Opens non blocking read file descriptor on console Use instance method .close to close file descriptor Use instance methods .getline to read & .put to write to console Needs os module

bs

max buffer size for each read, defaults to 256

Type
int

fd

file descriptor for console

Type
int

opened

True means .fd opened, False means .fd closed

Type
bool

rxbs

of received characters (bytes)

Type
bytearray

reopen()

closes and reopens .fd, sets .opened

close()

closes .fd unsets .opened

get()

returns chars including newline but no more than bs characters

put()

puts characters

Hidden:
._line is bytearray of line buffer

MaxBufSize = 256

open(*port=''*)

Opens fd on terminal console in non blocking mode.

port is the serial port device path name or if '' then use os.ctermid() which returns path name of console usually '/dev/tty'

os.O_NONBLOCK makes non blocking io os.O_RDWR allows both read and write. os.O_NOCTTY don't make this the controlling terminal of the process O_NOCTTY is only for cross platform portability BSD never makes it the controlling terminal

Don't use print at same time since it will mess up non blocking reads.

Works in both canonical and non-canonical input mode. In canonical mode, no chars are available to read until eol newline is entered and eol is included in the read characters.

It appears that canonical mode is the default for fd console os.ctermid(). For other serial port fds the characters may be available immediately.

To debug use os.isatty(fd) which returns True if the file descriptor fd is open and connected to a tty-like device, else False.

reopen()

Idempotently opens console

close()

Closes fd.

put(*data=b'\n'*)

Writes data bytes to console and return number of bytes from data written.

get(*bs=None*)

Gets nonblocking line of bytes from console of up to bs characters including eol newline if in bs characters otherwise must repeat get until a newline appears.

Returns empty string if no characters available else returns line. Works in both canonical and non-canonical mode In canonical mode, no chars are available to read until eol newline is entered and eol is included in the read characters.

Strips eol newline before returning line.

class hio.core.serial.serialing.**ConsoleDoer**(*console*, ***kwa*)

Bases: *hio.base.doing.Doer*

Basic Console Doer. Wraps console in doer context so opens and closes console

To test in WingIde must configure Debug I/O to use external console See Doer for inherited attributes, properties, and methods.

.console is serial Console instance

enter()

exit()

class hio.core.serial.serialing.**EchoConsoleDoer**(*console*, *lines=None*, *txbs=None*, ***kwa*)

Bases: *hio.base.doing.Doer*

Basic Terminal Console IO to buffers. Echos input back to output

To test in WingIde must configure Debug I/O to use external console

See Doer for inherited attributes, properties, and methods.

.console is serial Console instance

enter()

recur(*tyme*)

exit()

class hio.core.serial.serialing.**Device**(*port=None*, *speed=9600*, *bs=1024*)

Class to manage non blocking IO on serial device port.

Opens non blocking read file descriptor on serial port Use instance method close to close file descriptor Use instance methods get & put to read & write to serial device Needs os module

reopen(*port=None*, *speed=None*, *bs=None*)

Idempotently open serial device port Opens fd on serial port in non blocking mode.

port is the serial port device path name or if '' then use os.ctermid() which returns path name of console usually '/dev/tty'

os.O_NONBLOCK makes non blocking io os.O_RDWR allows both read and write. os.O_NOCTTY don't make this the controlling terminal of the process O_NOCTTY is only for cross platform portability BSD never makes it the controlling terminal

Don't use print and console at same time since it will mess up non blocking reads.

The input mode, canonical or noncanonical, is controlled by the ICANON flag see termios module.

Raw mode

def setraw(fd, when=TCSAFLUSH):
Put terminal into a raw mode. mode = tcgetattr(fd) mode[IFLAG] = mode[IFLAG] & ~(BRKINT | ICRNL | INPCK | ISTRIP | IXON) mode[OFLAG] = mode[OFLAG] & ~(OPOST) mode[CFLAG] = mode[CFLAG] & ~(CSIZE | PARENB) mode[CFLAG] = mode[CFLAG] | CS8 mode[LFLAG] = mode[LFLAG] & ~(ECHO | ICANON | IEXTEN | ISIG) mode[CC][VMIN] = 1 mode[CC][VTIME] = 0 tcsetattr(fd, when, mode)

set up raw mode / no echo / binary cflag **|= (TERMIOS.CLOCAL|TERMIOS.CREAD) lflag &= ~(TERMIOS.ICANON|TERMIOS.ECHO|TERMIOS.ECHOE|TERMIOS.ECHOK|TERMIOS.ECHONL|**

TERMIOS.ISIG|TERMIOS.IEXTEN) #|TERMIOS.ECHOPRT

for flag in ('ECHOCTL', 'ECHOKE'): # netbsd workaround for Erk

if hasattr(TERMIOS, flag):
lflag &= ~getattr(TERMIOS, flag)

oflag &= ~(TERMIOS.OPOST) iflag &= ~(TERMIOS.INLCR|TERMIOS.IGNCR|TERMIOS.ICRNL|TERMIOS.IGNBRK)
if hasattr(TERMIOS, 'IUCLC'):

iflag &= ~TERMIOS.IUCLC

if hasattr(TERMIOS, 'PARMRK'):
iflag &= ~TERMIOS.PARMRK

close()

Closes fd.

receive()

Reads nonblocking characters from serial device up to bs characters Returns empty bytes if no characters available else returns all available. In canonical mode no chars are available until newline is entered.

send(*data=b'\n'*)

Writes data bytes to serial device port. Returns number of bytes sent

class hio.core.serial.serialing.**Serial**(*port=None*, *speed=9600*, *bs=1024*)

Class to manage non blocking IO on serial device port using pyserial

Opens non blocking read file descriptor on serial port Use instance method close to close file descriptor Use instance methods get & put to read & write to serial device Needs os module

reopen(*port=None*, *speed=None*, *bs=None*)

Opens fd on serial port in non blocking mode.

port is the serial port device path name or if None then use os.ctermid() which returns path name of console usually '/dev/tty'

close()

Closes .serial

receive()

Reads nonblocking characters from serial device up to bs characters Returns empty bytes if no characters available else returns all available. In canonical mode no chars are available until newline is entered.

send(*data=b'\n'*)

Writes data bytes to serial device port. Returns number of bytes sent

class hio.core.serial.serialing.**Driver**(*name=''*, *uid=0*, *port=None*, *speed=9600*, *bs=1024*, *server=None*)

Nonblocking Serial Device Port Driver

serviceReceives()

Service receives until no more

clearRxbs()

Clear .rxbs

scan(*start*)

Returns offset of given start byte in self.rxbs Returns None if start is not given or not found If strip then remove any bytes before offset

send(*data*)

Handle one tx data

tx(*data*)

Queue data onto .txbs

serviceSends()

Service .txbs

service()

Sevice receives and sends

Package Contents

Classes

Console	Class to manage non blocking interactive I/O on serial console

class hio.core.serial.**Console**(*bs=None*)

Class to manage non blocking interactive I/O on serial console

Opens non blocking read file descriptor on console Use instance method .close to close file descriptor Use instance methods .getline to read & .put to write to console Needs os module

bs

max buffer size for each read, defaults to 256

Type
int

fd

file descriptor for console

Type
int

opened

True means .fd opened, False means .fd closed

Type
bool

rxbs

of received characters (bytes)

Type
bytearray

reopen()

closes and reopens .fd, sets .opened

close()

closes .fd unsets .opened

get()

returns chars including newline but no more than bs characters

put()

puts characters

Hidden:
._line is bytearray of line buffer

MaxBufSize = 256

open(*port=''*)

Opens fd on terminal console in non blocking mode.

port is the serial port device path name or if '' then use os.ctermid() which returns path name of console usually '/dev/tty'

os.O_NONBLOCK makes non blocking io os.O_RDWR allows both read and write. os.O_NOCTTY don't make this the controlling terminal of the process O_NOCTTY is only for cross platform portability BSD never makes it the controlling terminal

Don't use print at same time since it will mess up non blocking reads.

Works in both canonical and non-canonical input mode. In canonical mode, no chars are available to read until eol newline is entered and eol is included in the read characters.

It appears that canonical mode is the default for fd console os.ctermid(). For other serial port fds the characters may be available immediately.

To debug use os.isatty(fd) which returns True if the file descriptor fd is open and connected to a tty-like device, else False.

reopen()

Idempotently opens console

close()

Closes fd.

put(*data=b'\n'*)

Writes data bytes to console and return number of bytes from data written.

get(*bs=None*)

Gets nonblocking line of bytes from console of up to bs characters including eol newline if in bs characters otherwise must repeat get until a newline appears.

Returns empty string if no characters available else returns line. Works in both canonical and non-canonical mode In canonical mode, no chars are available to read until eol newline is entered and eol is included in the read characters.

Strips eol newline before returning line.

hio.core.tcp

hio.core.tcp Package

Submodules

hio.core.tcp.clienting

hio.core.tcp.clienting Module

Module Contents

Classes

Client	Nonblocking TCP Socket Client Class.
ClientTls	Outgoer with Nonblocking TLS/SSL support
ClientDoer	Basic TCP Client

Functions

openClient([cls])	Wrapper to create and open TCP Client instances

Attributes

logger

hio.core.tcp.clienting.**logger**

hio.core.tcp.clienting.**openClient**(*cls=None*, ***kwa*)

Wrapper to create and open TCP Client instances When used in with statement block, calls .close() on exit of with block

Parameters
instance (*cls is Class instance of subclass*) –

Usage:

with openClient() as client0:
client0.accept()

with openClient(cls=ClientTls) as client0:
client0.accept()

class hio.core.tcp.clienting.**Client**(*tymeout=None*, *ha=None*, *host='127.0.0.1'*, *port=56000*, *reconnectable=None*, *bs=8096*, *txbs=None*, *rxbs=None*, *wl=None*, ***kwa*)

Bases: *hio.base.tyming.Tymee*

Nonblocking TCP Socket Client Class.

See tyming.Tymee for inherited attributes, properties, and methods

Attributes:

Properties:

Methods:

property host

Property that returns host in .ha duple

property port

Property that returns port in .ha duple

property accepted

Property that returns accepted state of TCP socket

property connected

Property that returns connected state of TCP socket Non-tls tcp is connected when accepted

Tymeout = 0.0

Reconnectable = False

wind(*tymth*)

Inject new tymist.tymth as new ._tymth. Changes tymist.tyme base. Updates winds .tymer .tymth

reinitHostPort(*ha=None*, *hostname='127.0.0.1'*, *port=56000*)

Reinit self.ha and self.hostname from ha = (host, port) or hostname port self.ha is of form (host, port) where host is either dns name or ip address self.hostname is hostname as dns name host eventually is host ip address output from normalizeHost()

actualBufSizes()

Returns duple of the the actual socket send and receive buffer size (send, receive)

open()

Opens connection socket in non blocking mode.

if socket not closed properly, binding socket gets error
OSError: (48, 'Address already in use')

reopen()

Idempotently opens socket

shutdown(*how=socket.SHUT_RDWR*)

Shutdown connected socket .cs

shutdownSend()

Shutdown send on connected socket .cs

shutdownReceive()

Shutdown receive on connected socket .cs

close()

Shutdown and close connected socket .cs

refresh()

Restart timer

accept()

Attempt nonblocking acceptance connect to .ha Returns True if successful Returns False if not so try again later

connect()

Attempt nonblocking connect to .ha Returns True if successful Returns False if not so try again later For non-TLS tcp connect is done when accepted This is placeholder for subclass Tls

serviceConnect()

Service connection attempt If not already connected make a nonblocking attempt Returns .connected

receive()

Perform non blocking receive from connected socket .cs

If no data then returns None If connection closed then returns empty Otherwise returns data data is string in python2 and bytes in python3

serviceReceives()

Service receives until no more

serviceReceiveOnce()

Retrieve from server only one reception

clearRxbs()

Clear .rxbs

send(*data*)

Perform non blocking send on connected socket .cs. Return number of bytes sent data is string in python2 and bytes in python3

tx(*data*)

Copy data onto .txbs, .extend copies data.

serviceSends()

Service sends (transmits) of data in .txbs bytearray Attempt to send all of .txbs. Delete what is actually sent.

service()

Service connect, txbs, and receives.

class hio.core.tcp.clienting.**ClientTls**(*context=None*, *version=None*, *certify=None*, *hostify=None*, *certedhost=''*, *keypath=None*, *certpath=None*, *cafilepath=None*, ***kwa*)

Bases: *Client*

Outgoer with Nonblocking TLS/SSL support Nonblocking TCP Socket Client Class.

Attributes:

Properties:

Methods:

property connected

Property that returns connected state of TCP socket TLS tcp is connected when accepted and handshake completed

close()

Shutdown and close connected socket .cs

wrap()

Wrap socket .cs in ssl context

handshake()

Attempt nonblocking ssl handshake to .ha Returns True if successful Returns False if not so try again later

connect()

Attempt nonblocking connect to .ha Returns True if successful Returns False if not so try again later Connected when both accepted connection and TLS handshake complete

receive()

Perform non blocking receive from connected socket .cs

If no data then returns None If connection closed then returns '' Otherwise returns data data is string in python2 and bytes in python3

send(*data*)

Perform non blocking send on connected socket .cs. Return number of bytes sent data is string in python2 and bytes in python3

class hio.core.tcp.clienting.**ClientDoer**(*client*, ***kwa*)

Bases: *hio.base.doing.Doer*

Basic TCP Client

See Doer for inherited attributes, properties, and methods.

.client is TCP Client instance

wind(*tymth*)

Inject new tymist.tymth as new ._tymth. Changes tymist.tyme base. Updates winds .tymer .tymth

enter()

recur(*tyme*)

exit()

hio.core.tcp.serving

hio.core.tcp.serving Module

Accepter listens and accepts incoming TCP socket connections Server is subclass of Acceptor Server creates Remoters Remoter is accepted incoming socket connection

ServerTls is subclass of Server RemoterTls is subclass of Remoter

Module Contents

Classes

Acceptor	Acceptor Base Class for Server.
Server	Nonblocking TCP Socket Server Class.
ServerTls	Server with Nonblocking TLS/SSL support
Remoter	Class to service an incoming nonblocking TCP connection from a remote client.
RemoterTls	Class to service an incoming nonblocking TCP/TLS connection from a remote client.
ServerDoer	Basic TCP Server Doer
EchoServerDoer	Echo TCP Server

Functions

openServer([cls])	Wrapper to create and open TCP Server instances
initServerContext([context, version, certify, ...])	Initialize and return context for TLS Server

Attributes

logger

hio.core.tcp.serving.**logger**

hio.core.tcp.serving.**openServer**(*cls=None*, ***kwa*)

Wrapper to create and open TCP Server instances When used in with statement block, calls .close() on exit of with block

Parameters
instance (*cls is Class instance of subclass*) –

Usage:

with openServer() as server0:
server0.

with openServer(cls=ServerTls) as server0:
server0.

class hio.core.tcp.serving.**Acceptor**(*ha=None*, *bs=8096*, ***kwa*)

Bases: *hio.base.tyming.Tymee*

Acceptor Base Class for Server. Nonblocking TCP Socket Acceptor Class. Listen socket for incoming TCP connections

See tyming.Tymee for inherited attributes, properties, and methods

.ha is

host = “” or “0.0.0.0” means listen on all interfaces

Type
host,port) duple (two tuple

.eha is normalized

as external facing address for TLS context

Type
host, port

.bs is buffer size

.ss is server listen socket for incoming accept requests

.axes is deque of accepte connection duples

Type
ca, cs

.opened is boolean, True if listen socket .ss opened. False otherwise

actualBufSizes()

Returns duple of the the actual socket send and receive buffer size (send, receive)

open()

Opens binds listen socket in non blocking mode.

if socket not closed properly, binding socket gets error
OSError: (48, ‘Address already in use’)

reopen()

Idempotently opens listen socket

close()

Closes listen socket.

accept()

Accept new connection nonblocking Returns duple (cs, ca) of connected socket and connected host address Otherwise if no new connection returns (None, None)

serviceAccepts()

Service any accept requests Adds to .cxes dict key by ca

class hio.core.tcp.serving.**Server**(*ha=None*, *host=''*, *port=56000*, *tymeout=None*, *wl=None*, ***kwa*)

Bases: *Acceptor*

Nonblocking TCP Socket Server Class. Listen socket for incoming TCP connections that generates Remoter sockets for accepted connections

See tyming.Tymee for inherited attributes, properties, and methods

Inherited Attributes:

.ha is (host,port) duple (two tuple)
host = “” or “0.0.0.0” means listen on all interfaces

.eha is normalized (host, port) duple for incoming TLS connections
as external facing address for TLS context

.bs is buffer size .ss is server listen socket for incoming accept requests .axes is deque of accepte connection duples (ca, cs) .opened is boolean, True if listen socket .ss opened. False otherwise

.tymeout is tymeout in seconds for connection refresh

.wl is WireLog instance if any

.ixes is dict of incoming connections indexed by remote

Type
host, port

Tymeout = 1.0

wind(*tymth*)

Inject new tymist.tymth as new ._tymth. Changes tymist.tyme base. Updates winds .tymer .tymth

serviceAxes()

Service axes

For each newly accepted connection in .axes create Remoter and add to .ixes keyed by ca

serviceConnects()

Service connects is method name to be used

shutdownIx(*ca*, *how=socket.SHUT_RDWR*)

Shutdown remoter given by connection address ca

shutdownSendIx(*ca*)

Shutdown send on remoter given by connection address ca

shutdownReceiveIx(*ca*)

Shutdown send on remoter given by connection address ca

closeIx(*ca*)

Shutdown and close remoter given by connection address ca

closeAllIx()

Shutdown and close all remoter connections

close()

Close all sockets

removeIx(*ca*, *close=True*)

Remove remoter given by connection address ca

serviceReceivesIx(*ca*)

Service receives for remoter by connection address ca

serviceReceivesAllIx()

Service receives for all remoters in .ixes

transmitIx(*data*, *ca*)

Queue data onto .txbs for remoter given by connection address ca

serviceSendsAllIx()

Service transmits for all remoters in .ixes

service()

Service connects and service receives and sends for all ix.

hio.core.tcp.serving.**initServerContext**(*context=None*, *version=None*, *certify=None*, *keypath=None*, *certpath=None*, *cafilepath=None*)

Initialize and return context for TLS Server IF context is None THEN create a context

IF version is None THEN create context using ssl library default ELSE create context with version

If certify is not None then use certify value provided Otherwise use default

context = context object for tls/ssl If None use default version = ssl protocol version If None use default certify = cert requirement If None use default

ssl.CERT_NONE = 0 ssl.CERT_OPTIONAL = 1 ssl.CERT_REQUIRED = 2

keypath = pathname of local server side PKI private key file path
If given apply to context

certpath = pathname of local server side PKI public cert file path
If given apply to context

cafilepath = Cert Authority file path to use to verify client cert
If given apply to context

class hio.core.tcp.serving.**ServerTls**(*context=None*, *version=None*, *certify=None*, *keypath=None*, *certpath=None*, *cafilepath=None*, ***kwa*)

Bases: *Server*

Server with Nonblocking TLS/SSL support Nonblocking TCP Socket Server Class. Listen socket for incoming TCP connections RemoterTLS sockets for accepted connections

See tyming.Tymee for inherited attributes, properties, and methods

Inherited Attributes:

.ha is (host,port) duple (two tuple)
host = "" or "0.0.0.0" means listen on all interfaces

.eha is normalized (host, port) duple for incoming TLS connections
as external facing address for TLS context

.bs is buffer size .ss is server listen socket for incoming accept requests .axes is deque of accepte connection duples (ca, cs) .opened is boolean, True if listen socket .ss opened. False otherwise .timeout is timeout in seconds for connection refresh .wl is WireLog instance if any .ixes is dict of incoming connections indexed by remote (host, port) duple

.context is TLS context instance

.version is TLS version

.certify is boolean, True to client certify, False otherwise

.keypath is path to key file

.certpath is path to cert file

.cafilepath is path to ca file

serviceAxes()

Service accepteds

For each new accepted connection create RemoterTLS and add to .cxes Not Handshaked

serviceCxes()

Service handshakes for every remoter in .cxes If successful move to .ixes

serviceConnects()

Service accept and handshake attempts If not already accepted and handshaked Then

make nonblocking attempt

For each successful handshaked add to .ixes Returns handshakeds

class hio.core.tcp.serving.**Remoter**(*ha*, *ca*, *cs*, *tymeout=None*, *refreshable=True*, *bs=8096*, *wl=None*, ***kwa*)

Bases: *hio.base.tyming.Tymee*

Class to service an incoming nonblocking TCP connection from a remote client. Should only be used by an Acceptor subclass such as Server

Tymeout = 0.0

wind(*tymth*)

Inject new tymist.tymth as new ._tymth. Changes tymist.tyme base. Updates winds .tymer .tymth

shutdown(*how=socket.SHUT_RDWR*)

Shutdown connected socket .cs

shutdownSend()

Shutdown send on connected socket .cs

shutdownReceive()

Shutdown receive on connected socket .cs

close()

Shutdown and close connected socket .cs

refresh()

Restart tymer

receive()

Perform non blocking receive on connected socket .cs

If no data then returns None If connection closed then returns '' Otherwise returns data

data is string in python2 and bytes in python3

serviceReceives()

Service receives until no more

serviceReceiveOnce()

Retrieve from server only one reception

clearRxbs()

Clear .rxbs

send(*data*)

Perform non blocking send on connected socket .cs. Return number of bytes sent

data is string in python2 and bytes in python3

tx(*data*)

Queue data onto .txbs

serviceSends()

Service transmits For each tx if all bytes sent then keep sending until partial send or no more to send If partial send reattach and return

class hio.core.tcp.serving.**RemoterTls**(*context=None*, *version=None*, *certify=None*, *keypath=None*, *certpath=None*, *cafilepath=None*, ***kwa*)

Bases: *Remoter*

Class to service an incoming nonblocking TCP/TLS connection from a remote client. Should only be used from Acceptor subclass Provides nonblocking TLS/SSL support

connected

True means TLS handshake completed False otherwise

Type
bool

aborted

True means client aborted TLS handshake False otherwise

Type
bool

close()

Shutdown and close connected socket .cs

wrap()

Wrap socket .cs in ssl context

handshake()

Attempt nonblocking ssl handshake to .ha Sets .connected when complete successfully Sets .aborted when client terminates prematurely Keep trying while not connected and not aborted

receive()

Perform non blocking receive on connected socket .cs

If no data then returns None If connection closed then returns '' Otherwise returns data

data is string in python2 and bytes in python3

send(*data*)

Perform non blocking send on connected socket .cs. Return number of bytes sent

data is string in python2 and bytes in python3

class hio.core.tcp.serving.**ServerDoer**(*server*, ***kwa*)

Bases: *hio.base.doing.Doer*

Basic TCP Server Doer

See Doer for inherited attributes, properties, and methods.

.server is TCP Server instance

Properties:

wind(*tymth*)

Inject new tymist.tymth as new ._tymth. Changes tymist.tyme base. Updates winds .tymer .tymth

enter()

recur(*tyme*)

exit()

class hio.core.tcp.serving.**EchoServerDoer**(*server*, ***kwa*)

Bases: *ServerDoer*

Echo TCP Server Just echoes back to client whatever it receives from client

See Doer for inherited attributes, properties, and methods.

Inherited Attributes:
.server is TCP Server instance

enter()

recur(*tyme*)

exit()

hio.core.tcp.tcping

hio.core.tcping Module

Module Contents

Classes

Peer	Nonblocking TCP Socket Peer Class.

class hio.core.tcp.tcping.**Peer**(***kwa*)

Bases: *hio.core.tcp.serving.Server*

Nonblocking TCP Socket Peer Class. Supports both incoming and outgoing connections.

Package Contents

Classes

Client	Nonblocking TCP Socket Client Class.
ClientTls	Outgoer with Nonblocking TLS/SSL support
ClientDoer	Basic TCP Client
Server	Nonblocking TCP Socket Server Class.
ServerTls	Server with Nonblocking TLS/SSL support
Remoter	Class to service an incoming nonblocking TCP connection from a remote client.
ServerDoer	Basic TCP Server Doer
EchoServerDoer	Echo TCP Server

Functions

openClient([cls])	Wrapper to create and open TCP Client instances
openServer([cls])	Wrapper to create and open TCP Server instances

hio.core.tcp.**openClient**(*cls=None*, ***kwa*)

Wrapper to create and open TCP Client instances When used in with statement block, calls .close() on exit of with block

Parameters
instance (*cls is Class instance of subclass*) –

Usage:

with openClient() as client0:
client0.accept()

with openClient(cls=ClientTls) as client0:
client0.accept()

class hio.core.tcp.**Client**(*tymeout=None*, *ha=None*, *host='127.0.0.1'*, *port=56000*, *reconnectable=None*, *bs=8096*, *txbs=None*, *rxbs=None*, *wl=None*, ***kwa*)

Bases: hio.base.tyming.Tymee

Nonblocking TCP Socket Client Class.

See tyming.Tymee for inherited attributes, properties, and methods

Attributes:

Properties:

Methods:

property host

Property that returns host in .ha duple

property port

Property that returns port in .ha duple

property accepted

Property that returns accepted state of TCP socket

property connected

Property that returns connected state of TCP socket Non-tls tcp is connected when accepted

Tymeout = 0.0

Reconnectable = False

wind(*tymth*)

Inject new tymist.tymth as new ._tymth. Changes tymist.tyme base. Updates winds .tymer .tymth

reinitHostPort(*ha=None*, *hostname='127.0.0.1'*, *port=56000*)

Reinit self.ha and self.hostname from ha = (host, port) or hostname port self.ha is of form (host, port) where host is either dns name or ip address self.hostname is hostname as dns name host eventually is host ip address output from normalizeHost()

actualBufSizes()

Returns duple of the the actual socket send and receive buffer size (send, receive)

open()

Opens connection socket in non blocking mode.

if socket not closed properly, binding socket gets error
OSError: (48, 'Address already in use')

reopen()

Idempotently opens socket

shutdown(*how=socket.SHUT_RDWR*)

Shutdown connected socket .cs

shutdownSend()

Shutdown send on connected socket .cs

shutdownReceive()

Shutdown receive on connected socket .cs

close()

Shutdown and close connected socket .cs

refresh()

Restart timer

accept()

Attempt nonblocking acceptance connect to .ha Returns True if successful Returns False if not so try again later

connect()

Attempt nonblocking connect to .ha Returns True if successful Returns False if not so try again later For non-TLS tcp connect is done when accepted This is placeholder for subclass Tls

serviceConnect()

Service connection attempt If not already connected make a nonblocking attempt Returns .connected

receive()

Perform non blocking receive from connected socket .cs

If no data then returns None If connection closed then returns empty Otherwise returns data data is string in python2 and bytes in python3

serviceReceives()

Service receives until no more

serviceReceiveOnce()

Retrieve from server only one reception

clearRxbs()

Clear .rxbs

send(*data*)

Perform non blocking send on connected socket .cs. Return number of bytes sent data is string in python2 and bytes in python3

tx(*data*)

Copy data onto .txbs, .extend copies data.

serviceSends()

Service sends (transmits) of data in .txbs bytearray Attempt to send all of .txbs. Delete what is actually sent.

service()

Service connect, txbs, and receives.

class hio.core.tcp.**ClientTls**(*context=None*, *version=None*, *certify=None*, *hostify=None*, *certedhost=''*, *keypath=None*, *certpath=None*, *cafilepath=None*, ***kwa*)

Bases: *Client*

Outgoer with Nonblocking TLS/SSL support Nonblocking TCP Socket Client Class.

Attributes:

Properties:

Methods:

property connected

Property that returns connected state of TCP socket TLS tcp is connected when accepted and handshake completed

close()

Shutdown and close connected socket .cs

wrap()

Wrap socket .cs in ssl context

handshake()

Attempt nonblocking ssl handshake to .ha Returns True if successful Returns False if not so try again later

connect()

Attempt nonblocking connect to .ha Returns True if successful Returns False if not so try again later Connected when both accepted connection and TLS handshake complete

receive()

Perform non blocking receive from connected socket .cs

If no data then returns None If connection closed then returns '' Otherwise returns data data is string in python2 and bytes in python3

send(*data*)

Perform non blocking send on connected socket .cs. Return number of bytes sent data is string in python2 and bytes in python3

class hio.core.tcp.**ClientDoer**(*client*, ***kwa*)

Bases: *hio.base.doing.Doer*

Basic TCP Client

See Doer for inherited attributes, properties, and methods.

.client is TCP Client instance

wind(*tymth*)

Inject new tymist.tymth as new ._tymth. Changes tymist.tyme base. Updates winds .tymer .tymth

enter()

recur(*tyme*)

exit()

hio.core.tcp.**openServer**(*cls=None*, ***kwa*)

Wrapper to create and open TCP Server instances When used in with statement block, calls .close() on exit of with block

Parameters
instance (*cls is Class instance of subclass*) –

Usage:

with openServer() as server0:
server0.

with openServer(cls=ServerTls) as server0:
server0.

class hio.core.tcp.**Server**(*ha=None*, *host=''*, *port=56000*, *tymeout=None*, *wl=None*, ***kwa*)

Bases: Acceptor

Nonblocking TCP Socket Server Class. Listen socket for incoming TCP connections that generates Remoter sockets for accepted connections

See tyming.Tymee for inherited attributes, properties, and methods

Inherited Attributes:

.ha is (host,port) duple (two tuple)
host = "" or "0.0.0.0" means listen on all interfaces

.eha is normalized (host, port) duple for incoming TLS connections
as external facing address for TLS context

.bs is buffer size .ss is server listen socket for incoming accept requests .axes is deque of accepte connection duples (ca, cs) .opened is boolean, True if listen socket .ss opened. False otherwise

.tymeout is tymeout in seconds for connection refresh

.wl is WireLog instance if any

.ixes is dict of incoming connections indexed by remote

Type
host, port

Tymeout = 1.0

wind(*tymth*)

Inject new tymist.tymth as new ._tymth. Changes tymist.tyme base. Updates winds .tymer .tymth

serviceAxes()

Service axes

For each newly accepted connection in .axes create Remoter and add to .ixes keyed by ca

serviceConnects()

Service connects is method name to be used

shutdownIx(*ca*, *how=socket.SHUT_RDWR*)

Shutdown remoter given by connection address ca

shutdownSendIx(*ca*)

Shutdown send on remoter given by connection address ca

shutdownReceiveIx(*ca*)

Shutdown send on remoter given by connection address ca

closeIx(*ca*)

Shutdown and close remoter given by connection address ca

closeAllIx()

Shutdown and close all remoter connections

close()

Close all sockets

removeIx(*ca*, *close=True*)

Remove remoter given by connection address ca

serviceReceivesIx(*ca*)

Service receives for remoter by connection address ca

serviceReceivesAllIx()

Service receives for all remoters in .ixes

transmitIx(*data*, *ca*)

Queue data onto .txbs for remoter given by connection address ca

serviceSendsAllIx()

Service transmits for all remoters in .ixes

service()

Service connects and service receives and sends for all ix.

class hio.core.tcp.**ServerTls**(*context=None*, *version=None*, *certify=None*, *keypath=None*, *certpath=None*, *cafilepath=None*, ***kwa*)

Bases: *Server*

Server with Nonblocking TLS/SSL support Nonblocking TCP Socket Server Class. Listen socket for incoming TCP connections RemoterTLS sockets for accepted connections

See tyming.Tymee for inherited attributes, properties, and methods

Inherited Attributes:

.ha is (host,port) duple (two tuple)
host = "" or "0.0.0.0" means listen on all interfaces

.eha is normalized (host, port) duple for incoming TLS connections
as external facing address for TLS context

.bs is buffer size .ss is server listen socket for incoming accept requests .axes is deque of accepte connection duples (ca, cs) .opened is boolean, True if listen socket .ss opened. False otherwise .timeout is timeout in seconds for connection refresh .wl is WireLog instance if any .ixes is dict of incoming connections indexed by remote (host, port) duple

.context is TLS context instance

.version is TLS version

.certify is boolean, True to client certify, False otherwise

.keypath is path to key file

.certpath is path to cert file

.cafilepath is path to ca file

serviceAxes()

Service accepteds

For each new accepted connection create RemoterTLS and add to .cxes Not Handshaked

serviceCxes()

Service handshakes for every remoter in .cxes If successful move to .ixes

serviceConnects()

Service accept and handshake attempts If not already accepted and handshaked Then

make nonblocking attempt

For each successful handshaked add to .ixes Returns handshakeds

class hio.core.tcp.Remoter(*ha*, *ca*, *cs*, *tymeout=None*, *refreshable=True*, *bs=8096*, *wl=None*, ***kwa*)

Bases: *hio.base.tyming.Tymee*

Class to service an incoming nonblocking TCP connection from a remote client. Should only be used by an Acceptor subclass such as Server

Tymeout = 0.0

wind(*tymth*)

Inject new tymist.tymth as new ._tymth. Changes tymist.tyme base. Updates winds .tymer .tymth

shutdown(*how=socket.SHUT_RDWR*)

Shutdown connected socket .cs

shutdownSend()

Shutdown send on connected socket .cs

shutdownReceive()

Shutdown receive on connected socket .cs

close()

Shutdown and close connected socket .cs

refresh()

Restart tymer

receive()

Perform non blocking receive on connected socket .cs

If no data then returns None If connection closed then returns '' Otherwise returns data

data is string in python2 and bytes in python3

serviceReceives()

Service receives until no more

serviceReceiveOnce()

Retrieve from server only one reception

clearRxbs()

Clear .rxbs

send(*data*)

Perform non blocking send on connected socket .cs. Return number of bytes sent

data is string in python2 and bytes in python3

tx(*data*)

Queue data onto .txbs

serviceSends()

Service transmits For each tx if all bytes sent then keep sending until partial send or no more to send If partial send reattach and return

class hio.core.tcp.**ServerDoer**(*server*, ***kwa*)

Bases: *hio.base.doing.Doer*

Basic TCP Server Doer

See Doer for inherited attributes, properties, and methods.

.server is TCP Server instance

Properties:

wind(*tymth*)

Inject new tymist.tymth as new ._tymth. Changes tymist.tyme base. Updates winds .tymer .tymth

enter()

recur(*tyme*)

exit()

class hio.core.tcp.**EchoServerDoer**(*server*, ***kwa*)

Bases: *ServerDoer*

Echo TCP Server Just echoes back to client whatever it receives from client

See Doer for inherited attributes, properties, and methods.

Inherited Attributes:

.server is TCP Server instance

enter()

recur(*tyme*)

exit()

hio.core.udp

hio.core.udp Package

Submodules

hio.core.udp.udping

hio.core.udping Module

Module Contents

Classes

Peer	Class to manage non blocking I/O on UDP socket.

Attributes

logger
UDP_MAX_DATAGRAM_SIZE
UDP_MAX_SAFE_PAYLOAD
UDP_MAX_PACKET_SIZE

hio.core.udp.udping.**logger**

hio.core.udp.udping.**UDP_MAX_DATAGRAM_SIZE**

hio.core.udp.udping.**UDP_MAX_SAFE_PAYLOAD = 548**

hio.core.udp.udping.**UDP_MAX_PACKET_SIZE**

class hio.core.udp.udping.**Peer**(*ha=None*, *host=''*, *port=55000*, *bufsize=1024*, *wl=None*, *bcast=False*)

Bases: `object`

Class to manage non blocking I/O on UDP socket.

actualBufSizes()

Returns duple of the the actual socket send and receive buffer size (send, receive)

open()

Opens socket in non blocking mode.

if socket not closed properly, binding socket gets error
OSError: (48, 'Address already in use')

reopen()

Idempotently open socket

close()

Closes socket and logs if any

receive()

Perform non blocking read on socket.

returns tuple of form (data, sa) if no data then returns (b'',None) but always returns a tuple with two elements

send(*data*, *da*)

Perform non blocking send on socket.

data is string in python2 and bytes in python3 da is destination address tuple (destHost, destPort)

Submodules

hio.core.coring

hio.core.coring Module

Module Contents

Functions

normalizeHost(host)	Returns ip address host string in normalized dotted form or empty string
getDefaultHost()	Returns host ip address of default interface using netifaces
getDefaultBroadcast()	Returns broadcast ip address of default interface using netifaces
arpCreate(ether, host[, interface, temp])	Create arp entry for ethernet mac address ether at ip address host on interface
arpDelete(host[, interface])	Delete arp entry for ip address host on interface

hio.core.coring.**normalizeHost**(*host*)

Returns ip address host string in normalized dotted form or empty string converts host parameter which may be the dns name or ip address Prefers ipv4 addresses over ipv6 in that it will only return the ipv6 address if no ipv4 address equivalent is available

hio.core.coring.**getDefaultHost**()

Returns host ip address of default interface using netifaces

hio.core.coring.**getDefaultBroadcast**()

Returns broadcast ip address of default interface using netifaces

hio.core.coring.**arpCreate**(*ether*, *host*, *interface='en0'*, *temp=True*)

Create arp entry for ethernet mac address ether at ip address host on interface If temp is false then the entry is permanent otherwise its temporary

Assumes added /etc/sudoers entry to run arp with no password for user's group $ sudo visudo

Group to run arp as root with no password Cmnd_Alias ARP = /usr/sbin/arp %arp_group ALL=(ALL) NOPASSWD: ARP

hio.core.coring.**arpDelete**(*host*, *interface='en0'*)

Delete arp entry for ip address host on interface

Assumes added /etc/sudoers entry to run arp with no password for user's group $ sudo visudo

Group to run arp as root with no password Cmnd_Alias ARP = /usr/sbin/arp %arp_group ALL=(ALL) NOPASSWD: ARP

hio.core.wiring

hio.help.wiring module

Module Contents

Classes

WireLog	For debugging of non-blocking transports, provides log files or in memory
WireLogDoer	Basic WireLog Doer

Functions

openWL([cls, name, temp])	Context manager wrapper WireLog instances.

hio.core.wiring.**openWL**(*cls=None*, *name='test'*, *temp=True*, ***kwa*)

Context manager wrapper WireLog instances. Defaults to temporary wire logs. Context 'with' statements call .close on exit of 'with' block

Parameters

- **instance** (*cls is Class instance of subclass*) –
- **wirelogs** (*name is str name of wirelog instance for filename so can have multiple*) – at different paths thar each use different file name
- **Boolean** (*temp is*) – Otherwise open in persistent directory, do not clear on close
- **directory** (*True means open in temporary*) – Otherwise open in persistent directory, do not clear on close
- **close** (*clear on*) – Otherwise open in persistent directory, do not clear on close

Usage:

with openWL(name="bob") as wl:
wl.writeRx ....

with openWL(name="eve", cls=SubclassedWireLog)

class hio.core.wiring.**WireLog**(*rxed=True*, *txed=True*, *samed=False*, *filed=False*, *fmt=None*, *name='main'*, *temp=False*, *prefix=None*, *headDirPath=None*, *reopen=False*, *clear=False*)

For debugging of non-blocking transports, provides log files or in memory buffers for logging 'over the wire' network tx and rx packets as bytes

Attributes:

.rxed is Boolean True means log rx .txed is Boolean True means log tx .samed is Boolean True means log both rx and tx to same file or buffer .filed is Boolean True means log to file False means log to memory buffer .fmt is io write bytes printf style format string

Default is b'

%(dx)b %(who)b: %(data)b ' where:

who is src or dst for rx tx respectively dx is the io direction and will be set to either b'tx' or b'rx' and data is the actual io data as bytes

to write io data without direction who or line feeds use fmt= b'%(data)b'

.name is str used in file name .temp is Boolean True means use /tmp directory .prefix is str used as part of path prefix and formating .headDirPath is str used as head of path .tailDirpath is str used as tail of path .altTailDirPath is str used a alternate tail of path .dirPath is full directory path .rxl is rx log io file or io buffer .txl is tx log io file or io buffer .opened is Boolean, True means file is opened Otherwise False

```
Prefix = hio

HeadDirPath = /usr/local/var

TailDirPath = wirelogs

AltHeadDirPath = ~

TempHeadDir = /tmp

TempPrefix = test_

TempSuffix = _temp

Format = b'\n%(dx)b %(who)b:\n%(data)b\n'
```

reopen(*rxed=None*, *txed=None*, *samed=None*, *filed=None*, *fmt=None*, *name=None*, *temp=None*, *headDirPath=None*, *clear=False*)

Use or Create if not preexistent, directory path for file .path First closes .path if already opened. If clear is True then also clears .path before reopening

Parameters

- **ignore.** (*fmt is optional bytes printf format If None or unchanged then*) – Otherwise when creating io use .rxed if not provided
- **ignore.** – Otherwise when creating io use .txed if not provided
- **ignore.** – Otherwise when creating io use .same if not provided
- **ignore.** – Otherwise when creating io use .filed if not provided
- **ignore.** – Otherwise when creating io use .fmt if not provided
- **name** (*name is optional*) –

  if None or unchanged then ignore otherwise recreate path
  When recreating path, If not provided use .name

- **boolean** (*temp is optional*) –

  If None ignore Otherwise
  Assign to .temp If True then open in temporary directory and clear on close, If False then open persistent directory

- **database** (*headDirPath is optional str head directory pathname of main*) –

  if None or unchanged then ignore otherwise recreate path
  When recreating path, If not provided use default .HeadDirpath

- **closing** (*clear is Boolean True means clear .path when*) –

flush()

flush files if any and opened. A file flush only moves from program buffer to operating system buffer. A file fsync moves from operating system buffer to disk.

close(*clear=False*)

Close io logs. If clear or self.temp then remove directory at .dirPath :param clear is boolean: :param True means clear directory at .dirPath if any:

clearDirPath()

Remove logfile directory at .dirPath

readRx()

Returns rx string buffer value if .buffify else None

readTx()

Returns tx string buffer value if .buffify else None

writeRx(*data*, *who=b''*)

Write bytes data received from source host port address tuple,

writeTx(*data*, *who=b''*)

Write bytes data transmitted to destination address da,

class hio.core.wiring.**WireLogDoer**(*wl*, ***kwa*)

Bases: *hio.base.doing.Doer*

Basic WireLog Doer

Inherited Attributes:

.done is Boolean completion state:
True means completed Otherwise incomplete. Incompletion maybe due to close or abort.

.wl is WireLog subclass

Inherited Properties:
.tyme is float ._tymist.tyme, relative cycle or artificial time .tock is float, desired time in seconds between runs or until next run,

non negative, zero means run asap

Properties:

.wind injects ._tymist dependency

.__call__ makes instance callable

Appears as generator function that returns generator

.do is generator method that returns generator

.enter is enter context action method

.recur is recur context action method or generator method

.exit is exit context method

.close is close context method

.abort is abort context method

Hidden:

._tymist is Tymist instance reference ._tock is hidden attribute for .tock property

enter()

exit()

Package Contents

Classes

WireLog	For debugging of non-blocking transports, provides log files or in memory
WireLogDoer	Basic WireLog Doer

Functions

openWL([cls, name, temp])	Context manager wrapper WireLog instances.

hio.core.**openWL**(*cls=None*, *name='test'*, *temp=True*, ***kwa*)

Context manager wrapper WireLog instances. Defaults to temporary wire logs. Context 'with' statements call .close on exit of 'with' block

Parameters

- **instance** (*cls is Class instance of subclass*) –
- **wirelogs** (*name is str name of wirelog instance for filename so can have multiple*) – at different paths thar each use different file name
- **Boolean** (*temp is*) – Otherwise open in persistent directory, do not clear on close
- **directory** (*True means open in temporary*) – Otherwise open in persistent directory, do not clear on close
- **close** (*clear on*) – Otherwise open in persistent directory, do not clear on close

Usage:

with openWL(name="bob") as wl:
wl.writeRx ....

with openWL(name="eve", cls=SubclassedWireLog)

class hio.core.**WireLog**(*rxed=True*, *txed=True*, *samed=False*, *filed=False*, *fmt=None*, *name='main'*, *temp=False*, *prefix=None*, *headDirPath=None*, *reopen=False*, *clear=False*)

For debugging of non-blocking transports, provides log files or in memory buffers for logging 'over the wire' network tx and rx packets as bytes

Attributes:
.rxed is Boolean True means log rx .txed is Boolean True means log tx .samed is Boolean True means log both rx and tx to same file or buffer .filed is Boolean True means log to file False means log to memory buffer .fmt is io write bytes printf style format string

Default is b'

%(dx)b %(who)b: %(data)b ' where:

who is src or dst for rx tx respectively dx is the io direction and will be set to either b'tx' or b'rx' and data is the actual io data as bytes

to write io data without direction who or line feeds use fmt= b'%(data)b'

.name is str used in file name .temp is Boolean True means use /tmp directory .prefix is str used as part of path prefix and formating .headDirPath is str used as head of path .tailDirpath is str used as tail of path .altTailDirPath is str used a alternate tail of path .dirPath is full directory path .rxl is rx log io file or io buffer .txl is tx log io file or io buffer .opened is Boolean, True means file is opened Otherwise False

Prefix = hio

HeadDirPath = /usr/local/var

TailDirPath = wirelogs

AltHeadDirPath = ~

TempHeadDir = /tmp

TempPrefix = test_

TempSuffix = _temp

Format = b'\n%(dx)b %(who)b:\n%(data)b\n'

reopen(*rxed=None*, *txed=None*, *samed=None*, *filed=None*, *fmt=None*, *name=None*, *temp=None*, *headDirPath=None*, *clear=False*)

Use or Create if not preexistent, directory path for file .path First closes .path if already opened. If clear is True then also clears .path before reopening

Parameters

- **ignore.** (*fmt is optional bytes printf format If None or unchanged then*) – Otherwise when creating io use .rxed if not provided
- **ignore.** – Otherwise when creating io use .txed if not provided
- **ignore.** – Otherwise when creating io use .same if not provided
- **ignore.** – Otherwise when creating io use .filed if not provided

- **ignore.** – Otherwise when creating io use .fmt if not provided
- **name** (*name is optional*) –

  if None or unchanged then ignore otherwise recreate path
  When recreating path, If not provided use .name
- **boolean** (*temp is optional*) –

  If None ignore Otherwise
  Assign to .temp If True then open in temporary directory and clear on close, If False then open persistent directory
- **database** (*headDirPath is optional str head directory pathname of main*) –

  if None or unchanged then ignore otherwise recreate path
  When recreating path, If not provided use default .HeadDirpath
- **closing** (*clear is Boolean True means clear .path when*) –

flush()

flush files if any and opened. A file flush only moves from program buffer to operating system buffer. A file fsync moves from operating system buffer to disk.

close(*clear=False*)

Close io logs. If clear or self.temp then remove directory at .dirPath :param clear is boolean: :param True means clear directory at .dirPath if any:

clearDirPath()

Remove logfile directory at .dirPath

readRx()

Returns rx string buffer value if .buffify else None

readTx()

Returns tx string buffer value if .buffify else None

writeRx(*data*, *who=b''*)

Write bytes data received from source host port address tuple,

writeTx(*data*, *who=b''*)

Write bytes data transmitted to destination address da,

class hio.core.**WireLogDoer**(*wl*, ***kwa*)

Bases: *hio.base.doing.Doer*

Basic WireLog Doer

Inherited Attributes:

.done is Boolean completion state:
True means completed Otherwise incomplete. Incompletion maybe due to close or abort.

.wl is WireLog subclass

Inherited Properties:
.tyme is float ._tymist.tyme, relative cycle or artificial time .tock is float, desired time in seconds between runs or until next run,

non negative, zero means run asap

Properties:

.wind injects ._tymist dependency

.__call__ makes instance callable

Appears as generator function that returns generator

.do is generator method that returns generator

.enter is enter context action method

.recur is recur context action method or generator method

.exit is exit context method

.close is close context method

.abort is abort context method

Hidden:

._tymist is Tymist instance reference ._tock is hidden attribute for .tock property

enter()

exit()

hio.demo

hio.demo package

Demo applications that use hio

Subpackages

hio.demo.web

hio.demo.web package

Demo web applications that use hio

Submodules

hio.demo.web.demo_web

Demo web server for static files for client side web app

Module Contents

Functions

run()	Use hio http server

Attributes

logger

hio.demo.web.demo_web.**logger**

hio.demo.web.demo_web.**run**()

Use hio http server

hio.demo.web.demoing

Demo web server for static files for client side web app

Module Contents

hio.demo.web.demoing.**logger**

hio.help

hio.help package

Submodules

hio.help.decking

keri.help.decking module

Support for Deck class

Module Contents

Classes

Deck	Extends deque to support deque access convenience methods .push and .pull

class hio.help.decking.**Deck**(*iterable=None*, *maxlen=None*)

Bases: collections.deque

Extends deque to support deque access convenience methods .push and .pull to remove confusion about which side of the deque to use (left or right).

Extends deque with .push an .pull methods to support a different pattern for access. .push does not allow a value of None to be added to the Deck. This enables retrieval with .pull(emptive=True) which returns None when empty instead of raising IndexError. This allows use of the walrus operator on a pull to both assign and check for empty. For example:

deck.extend([False, "", []]) # falsy elements but not None stuff = [] while (x := deck.pull(emptive=True)) is not None:

stuff.append(x)

assert stuff == [False, "", []] assert not deck

Local methods: .push(x) = add x if x is not None to the right side of deque (like append) .pull(x) = remove and return element from left side of deque (like popleft)

Inherited methods from deque: .append(x) = add x to right side of deque .appendleft(x) = add x to left side of deque .clear() = clear all items from deque leaving it a length 0 .count(x) = count the number of deque elements equal to x. .extend(iterable) = append elements of iterable to right side .extendleft(iterable) = append elemets of iterable to left side

(this reverses iterable)

.pop() = remove and return element from right side
if empty then raise IndexError

.popleft() = remove and return element from left side
if empty then raise IndexError

.remove(x) = remove first occurence of x left to right
if not found raise ValueError

.rotate(n) = rotate n steps to right if neg rotate to left

Built in methods supported: len(d) reversed(d) copy.copy(d) copy.deepcopy(d) subscripts d[0] d[-1]

Attributes: .maxlen = maximum size of Deck or None if unbounded

__repr__()

Custome repr for Deck

push(*elem: Any*)

If not None, add elem to right side of deque, Otherwise ignore :param elem: element to be appended to deck (deque) :type elem: Any

pull(*emptive=False*)

Remove and return elem from left side of deque, If empty and emptive return None else raise IndexError

Parameters
emptive (*bool*) – True means return None instead of raise IndexError when attempt to pull False means normal behavior of deque

hio.help.helping

hio.help.helping module

Module Contents

Classes

NonStringIterable	Allows isinstance check for iterable that is not a string
NonStringSequence	Allows isinstance check for sequence that is not a string

Functions

copyfunc(f[, name])	Copy a function in detail.
attributize(genie)	Decorator function:
repack(n, seq[, default])	Repacks seq into a generator of len n and returns the generator.
just(n, seq[, default])	Returns a generator of just the first n elements of seq and substitutes
nonStringIterable(obj)	Returns: (bool) True if obj is non-string iterable, False otherwise
nonStringSequence(obj)	Returns: (bool) True if obj is non-string sequence, False otherwise
isIterator(obj)	Returns True if obj is an iterator object, that is,
ocfn(path[, mode, perm])	Atomically open or create file from filepath.
dump(data, path)	Serialize data dict and write to file given by path where serialization is
load(path)	Return data read from file path as dict

hio.help.helping.**copyfunc**(*f*, *name=None*)

Copy a function in detail. To change name of func provide name parameter

functools to update_wrapper assigns and updates following attributes WRAPPER_ASSIGNMENTS = (‘__module__’, ‘__name__’, ‘__qualname__’, ‘__doc__’,

‘__annotations__’)

WRAPPER_UPDATES = (‘__dict__’,) Based on https://stackoverflow.com/questions/6527633/how-can-i-make-a-deepcopy-of-a-function-in-python https://stackoverflow.com/questions/13503079/how-to-create-a-copy-of-a-python-function

hio.help.helping.**attributize**(*genie*)

Decorator function:

Python generators do not support adding attributes. Adding support for attributes provides a way to pass information from a WSGI App that returns a generator to a WSGI server via the generator after the WSGI app has already started returning its body. The hio.http.Server WSGI server looks for the attributes ._status and ._headers and substitutes these if present. This allows a streaming WSGI App body iterator to later modify the headers and status taht will be returned before the body iterator began iterating. This is useful for web hooks or backend requests that are serviced by an async coroutine based WSGI app so that they may leverage the streaming support of standard WSGI but use a the coroutine based hio.http.Server as an async WSGI server.

This decorator takes a Duck Typing approach to decorating a generator function or method that returns a new function type instance that when called will return a generator like object that supports attributes. the new wrapped object acts like a generator but with attributes.

Parameters

genie (*generator function, generator method*) – is either a generator function that returns a generator object a generator method that returns a generator object

If genie is a generator function then a reference to its wrapper
is injected as the first positional argument to the orginal generator function. The convention is to use the parameter 'me' to refer to the injected reference to the wrapper.

If genie is a generator method, that is, its first parameter is 'self'
then a reference to its wrapper is injected as the second positional argument to the original generator method. the convention is to use the parameter 'me' to refer to the injected reference to the wrapper so as not to collide with the 'self' instance reference.

When wrapped the new type is AttributiveGenerator

Usage: # decorated generator function @attributize def bar(me, req=None, rep=None):

me._status = 400 # or copy from rep.status me._headers = odict(example="Hi") # or copy from rep.headers yield b"" yield b"" yield b"Hello There" return b"Goodbye"

gen = bar() msg = next(gen) # attributes set after first next gen._status gen._headers

decorated generator method class R:

@attributize def bar(self, me, req=None, rep=None):

self.name = "Peter" me._status = 400 # or copy from rep.status me._headers = odict(example="Hi") # or copy from rep.headers yield b"" yield b"" yield b"Hello There " + self.name.encode() return b"Goodbye"

r = R() gen = r.bar() msg = next(gen) # attributes set after first next gen._status gen._headers

use as function wrapper directly instead of as decorator def gf(me, x): # convention injected reference to attributed wrapper is 'me'

me.x = 5 me.y = 'a' cnt = 0 while cnt < x:

yield cnt cnt += 1

agf = attributize(gf) ag = agf(3) # body of gf is not run until first next call assert isIterator(ag) assert not hasattr(ag, 'x') assert not hasattr(ag, 'y') n = next(ag) # first run here which sets up attributes assert n == 0 assert hasattr(ag, 'x') assert hasattr(ag, 'y') assert ag.x == 5 assert ag.y =='a' n = next(ag) assert n == 1

Adding attributes to this injected reference makes them available as attributes of the resultant wrapper.

The HTTP WSGI server at hio.core.http.serving.Server uses an instance of hio.core.http.serving.Responder to generate the response for each WSGI request. The Responder instance checks its WSGI app generator for existence of the attributes ._status and ._headers. If so then it overrides its default response status with ._status and updates its default response headers with the headers in ._header. This allows a backend (webhook) to conveniently influence the response status and headers. The response body is returned by the generator itself.

Background: Unlike Python functions, Python generators do not support custom attributes and the generator locals dict at .gi_frame.f_locals dissappears once the generator is complete so its inconvenient.

Fixed attributes of generator objects. ['.__next__', '__iter__', 'close', 'gi_code', 'gi_frame', 'gi_running', 'gi_yieldfrom', 'send', 'throw']

hio.help.helping.**repack**(*n*, *seq*, *default=None*)

Repacks seq into a generator of len n and returns the generator. The purpose is to enable unpacking into n variables. The first n-1 elements of seq are returned as the first n-1 elements of the generator and any remaining elements are returned in a tuple as the last element of the generator default (None) is substituted for missing elements when len(seq) < n

Example:

x = (1, 2, 3, 4) tuple(repack(3, x)) (1, 2, (3, 4))

x = (1, 2, 3) tuple(repack(3, x)) (1, 2, (3,))

x = (1, 2) tuple(repack(3, x)) (1, 2, ())

x = (1,) tuple(repack(3, x)) (1, None, ())

x = () tuple(repack(3, x)) (None, None, ())

hio.help.helping.**just**(*n*, *seq*, *default=None*)

Returns a generator of just the first n elements of seq and substitutes default (None) for any missing elements. This guarantees that a generator of exactly n elements is returned. This is to enable unpacking into n varaibles

Example:

x = (1, 2, 3, 4) tuple(just(3, x)) (1, 2, 3) x = (1, 2, 3) tuple(just(3, x)) (1, 2, 3) x = (1, 2) tuple(just(3, x)) (1, 2, None) x = (1,) tuple(just(3, x)) (1, None, None) x = () tuple(just(3, x)) (None, None, None)

class hio.help.helping.**NonStringIterable**

Allows isinstance check for iterable that is not a string if isinstance(x, NonStringIterable):

classmethod **__subclasshook__**(*C*)

Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

class hio.help.helping.**NonStringSequence**

Allows isinstance check for sequence that is not a string if isinstance(x, NonStringSequence):

classmethod **__subclasshook__**(*C*)

Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

hio.help.helping.**nonStringIterable**(*obj*)

Returns: (bool) True if obj is non-string iterable, False otherwise

Another way that is less future proof return (hasattr(x, '__iter__') and not isinstance(x, (str, bytes)))

hio.help.helping.**nonStringSequence**(*obj*)

Returns: (bool) True if obj is non-string sequence, False otherwise

hio.help.helping.**isIterator**(*obj*)

Returns True if obj is an iterator object, that is,

has an __iter__ method has a __next__ method .__iter__ is callable and returns obj

Otherwise returns False

hio.help.helping.**ocfn**(*path*, *mode='r+'*, *perm=stat.S_IRUSR | stat.S_IWUSR*)

Atomically open or create file from filepath.

If file already exists, Then open file using openMode Else create file using write update mode If not binary Else

write update binary mode

Returns file object

If binary Then If new file open with write update binary mode

x = stat.S_IRUSR | stat.S_IWUSR 384 == 0o600 436 == octal 0664

hio.help.helping.**dump**(*data*, *path*)

Serialize data dict and write to file given by path where serialization is given by path's extension of either JSON, MsgPack, or CBOR for extension .json, .mgpk, or .cbor respectively

hio.help.helping.**load**(*path*)

Return data read from file path as dict file may be either json, msgpack, or cbor given by extension .json, .mgpk, or .cbor respectively Otherwise raise IOError

hio.help.hicting

hio.help.hicting module

Module Contents

Classes

Hict	Hict is a Case Insensitive Keyed Multiple valued dictionary like class that
Mict	Mict is a multiple valued dictionary like class that extends MultiDict.

class hio.help.hicting.**Hict**

Bases: multidict.CIMultiDict

Hict is a Case Insensitive Keyed Multiple valued dictionary like class that extends CIMultiDict and is used for HTTP headers which have case insentive labels. Insertion order of keys preserved. Associated with each key is a valuelist i.e. a list of values for that key.

https://multidict.readthedocs.io/en/stable/ CIMultiDict keys must be subclass of str no ints allowed In CIMultiDict:

.add(key,value) appends value to the valuelist at key

m["key"] = value replaces the valuelist at key with [value]

m["key"] returns the first added element of the valuelist at key

MultiDict methods access values in FIFO order Hict adds method to access values in LIFO order

Extended methods in Hict but not in CIMultiDict are:

nabone(key [,default]) get last value at key else default or KeyError nab(key [,default]) get last value at key else default or None naball(key [,default]) get all values inverse order else default or KeyError firsts() get all items where item value is first inserted value at key lasts() get all items where item value is last insterted value at key

__repr__()

nabone(*key*, **pa*, ***kwa*)

Usage:
.nabone(key [, default])

returns last value at key if key in dict else default raises KeyError if key not in dict and default not provided.

nab(*key*, **pa*, ***kwa*)

Usage:
.nab(key [, default])

returns last value at key if key in dict else default returns None if key not in dict and default not provided.

naball(*key*, **pa*, ***kwa*)

Usage:
.nabone(key [, default])

returns list of values at key if key in dict else default raises KeyError if key not in dict and default not provided.

firsts()

Returns list of (key, value) pair where each value is first value at key but with no duplicate keys. MultiDict .keys() returns a key for each duplicate value

lasts()

Returns list of (key, value) pairs where each value is last value at key but with no duplicate keys. MultiDict .keys() returns a key for each duplicate value

class hio.help.hicting.**Mict**

Bases: multidict.MultiDict

Mict is a multiple valued dictionary like class that extends MultiDict. Insertion order of keys preserved. Associated with each key is a valuelist i.e. a list of values for that key.

https://multidict.readthedocs.io/en/stable/ MultiDict keys must be subclass of str no ints allowed In MultiDict:

.add(key,value) appends value to the valuelist at key

m["key"] = value replaces the valuelist at key with [value]

m["key"] returns the first added element of the valuelist at key

MultiDict methods access values in FIFO order Mict adds methods to access values in LIFO order

Extended methods in Mict but not in MultiDict are:
nabone(key [,default]) get last value at key else default or KeyError nab(key [,default]) get last value at key else default or None naball(key [,default]) get all values inverse order else default or KeyError

__repr__()

nabone(*key*, **pa*, ***kwa*)

Usage:
.nabone(key [, default])

returns last value at key if key in dict else default raises KeyError if key not in dict and default not provided.

nab(*key*, **pa*, ***kwa*)

Usage:
.nab(key [, default])

returns last value at key if key in dict else default returns None if key not in dict and default not provided.

naball(*key*, **pa*, ***kwa*)

Usage:
.nabone(key [, default])

returns list of values at key if key in dict else default raises KeyError if key not in dict and default not provided.

firsts()

Returns list of (key, value) pair where each value is first value at key No duplicate keys

lasts()

Returns list of (key, value) pairs where each value is last value at key No duplicate keys

hio.help.ogling

hio.help.ogling module

Provides python stdlib logging module support

Module Contents

Classes

Ogler	Olger instances provide loggers as global logging facility

Functions

initOgler([level])	Initialize the ogler global instance once
openOgler([cls, name, temp])	Context manager wrapper Ogler instances.

hio.help.ogling.**initOgler**(*level=logging.CRITICAL*, ***kwa*)

Initialize the ogler global instance once Usage:

At top level of module in project # assign ogler as module global instance availabe at module-name.ogler ogler = hio.help.ogling.initOgler()

module is mypackage.help then ogler at mypackage.help.ogler

Critical is most severe to restrict logging by default

Parameters

- **None** (*force is Boolean True is to force reinit even if global ogler is not*) –
- **level** (*level is default logging*) –

This should be called in package .__init__ to insure that global ogler is defined by default. Users may then reset level and reopen log file if need be before calling ogler.getLoggers()

hio.help.ogling.**openOgler**(*cls=None*, *name='test'*, *temp=True*, ***kwa*)

Context manager wrapper Ogler instances. Defaults to temporary file logs. Context 'with' statements call .close on exit of 'with' block

Parameters

- **instance** (*cls is Class instance of subclass*) –
- **oglers** (*name is str name of ogler instance for filename so can have multiple*) – at different paths thar each use different log file directories
- **Boolean** (*temp is*) – Otherwise open in persistent directory, do not clear on close
- **directory** (*True means open in temporary*) – Otherwise open in persistent directory, do not clear on close
- **close** (*clear on*) – Otherwise open in persistent directory, do not clear on close

Usage:

with openOgler(name="bob") as ogler:
logger = ogler.getLogger() ....

with openOgler(name="eve", cls=SubclassedOgler)

class hio.help.ogling.**Ogler**(*name='main'*, *level=logging.ERROR*, *temp=False*, *prefix=None*, *headDirPath=None*, *reopen=False*, *clear=False*, *consoled=True*, *syslogged=True*, *filed=True*, *when='H'*, *interval=1*, *count=48*)

Olger instances provide loggers as global logging facility Only need one Ogler per application Uses python stdlib logging module, logging.getLogger(name). Multiple calls to .getLogger() with the same name will always return a

reference to the same Logger object.

name

usage specific component used in file name

Type
str

level

logging severity level

Type
int

temp

True means use /tmp directory

Type
bool

prefix

application specific path prefix and formating

Type
str

headDirPath

head of path

Type
str

dirPath

full directory path

Type
str

path

full file path

Type
str

opened

True means file is opened, False not opened

Type
bool

consoled

True means log to console (stderr), False do not

Type
bool

syslogged

True means log to syslog, False do not

Type
bool

filed

True means log to rotating file at .path, False do not

Type
bool

when

interval type for timed rotating file handler

Type
str

interval

length of interval of type when

Type
int

count

backup count number of backups to keep

Type
int

Prefix = hio

HeadDirPath = /usr/local/var

TailDirPath = logs

AltHeadDirPath = ~

TempHeadDir = /tmp

TempPrefix = test_

TempSuffix = _temp

reopen(*name=None*, *temp=None*, *headDirPath=None*, *clear=False*)

Use or Create if not preexistent, directory path .dirPath for file .path First closes .path if already opened. If clear is True then also clears .path before reopening

Parameters

- **name** (*name is optional*) –

  if None or unchanged then ignore otherwise recreate path
  When recreating path, If not provided use .name

- **boolean** (*temp is optional*) –

  If None ignore Otherwise
  Assign to .temp If True then open temporary directory, If False then open persistent directory

- **database** (*headDirPath is optional str head directory pathname of main*) –

  if None or unchanged then ignore otherwise recreate path
  When recreating path, If not provided use default .HeadDirpath

- **closing** (*clear is Boolean True means clear .path when*) –

close(*clear=False*)

Set .opened to False and remove directory at .path :param clear is boolean: :param True means clear directory:

clearDirPath()

Remove logfile directory at .dirPath

resetLevel(*name=__name__*, *level=None*, *globally=False*)

Resets the level of preexisting loggers to level. If level is None then use .level

getLogger(*name=__name__*, *level=None*)

Returns Basic Logger default is to name logger after module

hio.help.timing

hio.help.timing module

Module Contents

Classes

Timer	Class to manage real elaspsed time using time module.
MonoTimer	Class to manage real elaspsed time using time module but with monotonically

exception hio.help.timing.**TimerError**

Bases: *hio.hioing.HioError*

Generic Timer Errors Usage:

raise TimerError("error message")

exception hio.help.timing.**RetroTimerError**

Bases: *TimerError*

Error due to real time being retrograded before start time of timer Usage:

raise RetroTimerError("error message")

class hio.help.timing.**Timer**(*duration=0.0*, *start=None*, ***kwa*)

Bases: *hio.hioing.Mixin*

Class to manage real elaspsed time using time module. .. attribute:: ._start is start tyme in seconds

._stop is stop tyme in seconds

Properties:

.duration is float time duration in seconds of timer from ._start to ._stop .elaspsed is float time elasped in seconds since ._start .remaining is float time remaining in seconds until ._stop .expired is boolean, True if expired, False otherwise, i.e. time >= ._stop

.start() start timer at current time

.restart() = restart timer at last ._stop so no time lost

property duration

duration property getter, .duration = ._stop - ._start .duration is float duration tyme

property elapsed

elapsed time property getter, Returns elapsed time in seconds (fractional) since ._start.

property remaining

remaining time property getter, Returns remaining time in seconds (fractional) before ._stop.

property expired

Returns True if timer has expired, False otherwise. time.time() >= ._stop,

start(*duration=None*, *start=None*)

Starts Timer of duration secs at start time start secs.

If duration not provided then uses current duration If start not provided then starts at current time.time()

restart(*duration=None*)

Lossless restart of Timer at start = ._stop for duration if provided, Otherwise current duration. No time lost. Useful to extend Timer so no time lost

class hio.help.timing.**MonoTimer**(*duration=0.0*, *start=None*, *retro=True*)

Bases: *Timer*

Class to manage real elaspsed time using time module but with monotonically increating time guarantee in spite of system time being retrograded.

If the system clock is retrograded (moved back in time) while the timer is running then time.time() could move to before the start time. MonoTimer detects this retrograde and if retro is True then retrogrades the start and stop times back Otherwise it raises a TimerRetroError. MonoTimer is not able to detect a prograded clock (moved forward in time)

._start is start time in seconds

._stop is stop time in seconds

._last is last measured time in seconds with retrograde handling

.retro is boolean If True retrograde ._start and ._stop when time is retrograded.

Properties:

.duration is float time duration in seconds of timer from ._start to ._stop .elaspsed is float time elasped in seconds since ._start .remaining is float time remaining in seconds until ._stop .expired is boolean True if expired, False otherwise, i.e. time >= ._stop .latest is float latest measured time in seconds with retrograte handling

.start() = start timer at current time returns start time

.restart() = restart timer at last ._stop so no time lost, returns start time

property elapsed

elapsed time property getter, Returns elapsed time in seconds (fractional) since ._start.

property remaining

remaining time property getter, Returns remaining time in seconds (fractional) before ._stop.

property expired

Returns True if timer has expired, False otherwise. .latest >= ._stop,

property latest

latest measured time property getter, Returns latest measured time in seconds adjusted for retrograded system time.

Package Contents

Classes

Deck	Extends deque to support deque access convenience methods .push and .pull
Hict	Hict is a Case Insensitive Keyed Multiple valued dictionary like class that
Mict	Mict is a multiple valued dictionary like class that extends MultiDict.
Timer	Class to manage real elaspsed time using time module.
MonoTimer	Class to manage real elaspsed time using time module but with monotonically

Attributes

ogler

hio.help.**ogler**

class hio.help.**Deck**(*iterable=None*, *maxlen=None*)

Bases: collections.deque

Extends deque to support deque access convenience methods .push and .pull to remove confusion about which side of the deque to use (left or right).

Extends deque with .push an .pull methods to support a different pattern for access. .push does not allow a value of None to be added to the Deck. This enables retrieval with .pull(emptive=True) which returns None when empty instead of raising IndexError. This allows use of the walrus operator on a pull to both assign and check for empty. For example:

deck.extend([False, "", []]) # falsy elements but not None stuff = [] while (x := deck.pull(emptive=True)) is not None:

stuff.append(x)

assert stuff == [False, "", []] assert not deck

Local methods: .push(x) = add x if x is not None to the right side of deque (like append) .pull(x) = remove and return element from left side of deque (like popleft)

Inherited methods from deque: .append(x) = add x to right side of deque .appendleft(x) = add x to left side of deque .clear() = clear all items from deque leaving it a length 0 .count(x) = count the number of deque elements equal to x. .extend(iterable) = append elements of iterable to right side .extendleft(iterable) = append elemets of iterable to left side

(this reverses iterable)

.pop() = remove and return element from right side
if empty then raise IndexError

.popleft() = remove and return element from left side
if empty then raise IndexError

.remove(x) = remove first occurence of x left to right
if not found raise ValueError

.rotate(n) = rotate n steps to right if neg rotate to left

Built in methods supported: len(d) reversed(d) copy.copy(d) copy.deepcopy(d) subscripts d[0] d[-1]

Attributes: .maxlen = maximum size of Deck or None if unbounded

__repr__()

Custome repr for Deck

push(*elem: Any*)

If not None, add elem to right side of deque, Otherwise ignore :param elem: element to be appended to deck (deque) :type elem: Any

pull(*emptive=False*)

Remove and return elem from left side of deque, If empty and emptive return None else raise IndexError

Parameters
emptive (*bool*) – True means return None instead of raise IndexError when attempt to pull False means normal behavior of deque

class hio.help.**Hict**

Bases: multidict.CIMultiDict

Hict is a Case Insensitive Keyed Multiple valued dictionary like class that extends CIMultiDict and is used for HTTP headers which have case insentive labels. Insertion order of keys preserved. Associated with each key is a valuelist i.e. a list of values for that key.

https://multidict.readthedocs.io/en/stable/ CIMultiDict keys must be subclass of str no ints allowed In CIMultiDict:

.add(key,value) appends value to the valuelist at key

m["key"] = value replaces the valuelist at key with [value]

m["key"] returns the first added element of the valuelist at key

MultiDict methods access values in FIFO order Hict adds method to access values in LIFO order

Extended methods in Hict but not in CIMultiDict are:
nabone(key [,default]) get last value at key else default or KeyError nab(key [,default]) get last value at key else default or None naball(key [,default]) get all values inverse order else default or KeyError firsts() get all items where item value is first inserted value at key lasts() get all items where item value is last insterted value at key

__repr__()

nabone(*key*, **pa*, ***kwa*)

Usage:
.nabone(key [, default])

returns last value at key if key in dict else default raises KeyError if key not in dict and default not provided.

nab(*key*, **pa*, ***kwa*)

Usage:
.nab(key [, default])

returns last value at key if key in dict else default returns None if key not in dict and default not provided.

naball(*key*, **pa*, ***kwa*)

Usage:
.nabone(key [, default])

returns list of values at key if key in dict else default raises KeyError if key not in dict and default not provided.

firsts()

Returns list of (key, value) pair where each value is first value at key but with no duplicate keys. MultiDict .keys() returns a key for each duplicate value

lasts()

Returns list of (key, value) pairs where each value is last value at key but with no duplicate keys. MultiDict .keys() returns a key for each duplicate value

class hio.help.Mict

Bases: multidict.MultiDict

Mict is a multiple valued dictionary like class that extends MultiDict. Insertion order of keys preserved. Associated with each key is a valuelist i.e. a list of values for that key.

https://multidict.readthedocs.io/en/stable/ MultiDict keys must be subclass of str no ints allowed In MultiDict:

.add(key,value) appends value to the valuelist at key

m["key"] = value replaces the valuelist at key with [value]

m["key"] returns the first added element of the valuelist at key

MultiDict methods access values in FIFO order Mict adds methods to access values in LIFO order

Extended methods in Mict but not in MultiDict are:

nabone(key [,default]) get last value at key else default or KeyError nab(key [,default]) get last value at key else default or None naball(key [,default]) get all values inverse order else default or KeyError

__repr__()

nabone(*key*, **pa*, ***kwa*)

Usage:

.nabone(key [, default])

returns last value at key if key in dict else default raises KeyError if key not in dict and default not provided.

nab(*key*, **pa*, ***kwa*)

Usage:

.nab(key [, default])

returns last value at key if key in dict else default returns None if key not in dict and default not provided.

naball(*key*, **pa*, ***kwa*)

Usage:

.nabone(key [, default])

returns list of values at key if key in dict else default raises KeyError if key not in dict and default not provided.

firsts()

Returns list of (key, value) pair where each value is first value at key No duplicate keys

lasts()

Returns list of (key, value) pairs where each value is last value at key No duplicate keys

class hio.help.Timer(*duration=0.0*, *start=None*, ***kwa*)

Bases: *hio.hioing.Mixin*

Class to manage real elaspsed time using time module. .. attribute:: ._start is start tyme in seconds

._stop is stop tyme in seconds

Properties:

.duration is float time duration in seconds of timer from ._start to ._stop .elaspsed is float time elasped in seconds since ._start .remaining is float time remaining in seconds until ._stop .expired is boolean, True if expired, False otherwise, i.e. time >= ._stop

.start() start timer at current time

.restart() = restart timer at last ._stop so no time lost

property duration

duration property getter, .duration = ._stop - ._start .duration is float duration tyme

property elapsed

elapsed time property getter, Returns elapsed time in seconds (fractional) since ._start.

property remaining

remaining time property getter, Returns remaining time in seconds (fractional) before ._stop.

property expired

Returns True if timer has expired, False otherwise. time.time() >= ._stop,

start(*duration=None*, *start=None*)

Starts Timer of duration secs at start time start secs.
If duration not provided then uses current duration If start not provided then starts at current time.time()

restart(*duration=None*)

Lossless restart of Timer at start = ._stop for duration if provided, Otherwise current duration. No time lost. Useful to extend Timer so no time lost

class hio.help.**MonoTimer**(*duration=0.0*, *start=None*, *retro=True*)

Bases: *Timer*

Class to manage real elaspsed time using time module but with monotonically increating time guarantee in spite of system time being retrograded.

If the system clock is retrograded (moved back in time) while the timer is running then time.time() could move to before the start time. MonoTimer detects this retrograde and if retro is True then retrogrades the start and stop times back Otherwise it raises a TimerRetroError. MonoTimer is not able to detect a prograded clock (moved forward in time)

._start is start time in seconds

._stop is stop time in seconds

._last is last measured time in seconds with retrograde handling

.retro is boolean If True retrograde ._start and ._stop when time is retrograded.

Properties:

.duration is float time duration in seconds of timer from ._start to ._stop .elaspsed is float time elasped in seconds since ._start .remaining is float time remaining in seconds until ._stop .expired is boolean True if expired, False otherwise, i.e. time >= ._stop .latest is float latest measured time in seconds with retrograte handling

.start() = start timer at current time returns start time

.restart() = restart timer at last ._stop so no time lost, returns start time

property elapsed

elapsed time property getter, Returns elapsed time in seconds (fractional) since ._start.

property remaining

remaining time property getter, Returns remaining time in seconds (fractional) before ._stop.

property expired

Returns True if timer has expired, False otherwise. .latest >= ._stop,

property latest

latest measured time property getter, Returns latest measured time in seconds adjusted for retrograded system time.

exception hio.help.**TimerError**

Bases: *hio.hioing.HioError*

Generic Timer Errors Usage:

raise TimerError("error message")

exception hio.help.**RetroTimerError**

Bases: *TimerError*

Error due to real time being retrograded before start time of timer Usage:

raise RetroTimerError("error message")

3.1.2 Submodules

hio.__main__

hio package

Entrypoint module, in case you use *python -m hio*.

Why does this file exist, and why __main__? For more info, read:

- https://www.python.org/dev/peps/pep-0338/
- https://docs.python.org/3/using/cmdline.html#cmdoption-m

hio.cli

hio command line

Module that contains the command line app.

Why does this file exist, and why not put this in __main__?

You might be tempted to import things from __main__ later, but that will cause problems: the code will get executed twice:

- When you run *python -m keri* python will execute `__main__.py` as a script. That means there won't be any `hio.__main__` in `sys.modules`.
- When you import __main__ it will get executed again (as a module) because there's no `hio.__main__` in `sys.modules`.

Also see (1) from http://click.pocoo.org/5/setuptools/#setuptools-integration

Module Contents

Functions

main([args])

Attributes

parser

hio.cli.**parser**

hio.cli.**main**(*args=None*)

hio.daemon

hio daemon

Background Server Daemon for keri

Module Contents

Functions

main([args])

Attributes

parser

hio.daemon.**parser**

hio.daemon.**main**(*args=None*)

hio.hioing

hio.hioing module

Generic Constants and Classes Exception Classes

Module Contents

Classes

Mixin	Base class to enable consistent MRO for mixin multiple inheritance

Attributes

Versionage
Version
SEPARATOR
SEPARATOR_BYTES

hio.hioing.**Versionage**

hio.hioing.**Version**

hio.hioing.**SEPARATOR = Multiline-String**

```
1
```

hio.hioing.**SEPARATOR_BYTES**

exception hio.hioing.**HioError**

Bases: Exception

Base Class for hio exceptions

To use raise HioError("Error: message")

exception hio.hioing.**ValidationError**

Bases: *HioError*

Validation related errors Usage:

raise ValidationError("error message")

exception hio.hioing.**VersionError**

Bases: *ValidationError*

Bad or Unsupported Version

Usage:
raise VersionError("error message")

exception hio.hioing.**OglerError**

Bases: *HioError*

Error using or configuring Ogler

Usage:
raise OglerError("error message")

class hio.hioing.**Mixin**(**pa*, ***kwa*)

Base class to enable consistent MRO for mixin multiple inheritance Allows each subclass to call super(MixinSubClass, self).__init__(*pa, **kwa) So the __init__ propagates to common top of Tree https://medium.com/geekculture/cooperative-multiple-inheritance-in-python-practice-60e3ac5f91cc

3.1.3 Package Contents

Classes

Mixin	Base class to enable consistent MRO for mixin multiple inheritance

Attributes

__version__

hio.**__version__** = **0.6.9**

class hio.**Mixin**(**pa*, ***kwa*)

Base class to enable consistent MRO for mixin multiple inheritance Allows each subclass to call super(MixinSubClass, self).__init__(*pa, **kwa) So the __init__ propagates to common top of Tree https://medium.com/geekculture/cooperative-multiple-inheritance-in-python-practice-60e3ac5f91cc

exception hio.**HioError**

Bases: Exception

Base Class for hio exceptions

To use raise HioError("Error: message")

exception hio.**ValidationError**

Bases: *HioError*

Validation related errors Usage:

raise ValidationError("error message")

exception hio.**VersionError**

Bases: *ValidationError*

Bad or Unsupported Version

Usage:
raise VersionError("error message")

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

h

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W