HomeControl

Mar 26, 2020

Contents:

CHAPTER 1

dependencies module

1.1 Action Engine

ActionEngine for HomeControl

class homecontrol.dependencies.action_engine.**ActionEngine**(*item: homecontrol.dependencies.entity_types.Item*, *core*)

Bases: object

Holds available actions for an item

coroutine execute(*name: str*, **args*, ***kwargs*) → bool

Executes an action, optionally with parameters

homecontrol.dependencies.action_engine.**action**(*arg: Union[Callable, str]*) → Callable

Decorator to mark a coroutine as an action

```
>>> @action
>>> async def foo(): ...
```

```
>>> @action(name)
>>> async def foo(): ...
```

1.2 Configuration Manager

config_manager module

class homecontrol.dependencies.config_manager.**ConfigManager**(*cfg: dict*, *cfg_path: str*)

Bases: object

Manages the configuration with configuration domains

coroutine approve_domain_config(*domain: str*, *config: dict*, *initial: bool = True*) → dict
Returns an approved and validated version of config for a domain

get(*key*, *default=None*)
getter for self.cfg

coroutine register_domain(*domain: str*, *handler: object = None*, *schema: voluptuous.schema_builder.Schema = None*, *allow_reload: bool = False*, *default: dict = None*) → object
Registers a configuration domain

Objects can register themselves to their own configuration domain and subscribe to changes in the configuration

Parameters

- **domain** (*str*) – The configuration domain (A top-level key in config.yaml)
- **handler** (*object*) – The object subscribing to this domain. Methods it can implement are: - approve_configuration(domain, config) -> bool: - apply_new_configuration(domain, config) -> None: If not specified then your configuration domain will not be reloadable

allow_reload (bool): Allow reloading this configuration domain

schema (voluptuous.Schema): Schema to validate the configuration and to fill in defaults

default (dict): A default configuration

Returns A validated and approved version of the configuration

coroutine reload_config() → None
Reloads the configuration and updates where it can

1.3 Data Types

data types for HomeControl that can be translated to JSON for API usage

class homecontrol.dependencies.data_types.**Color**(*h: int*, *s: int*, *l: int*)
Bases: object

Representation for a color

dump() -> (<*class 'int'*>, <*class 'int'*>, <*class 'int'*>)
Dumps the Color into a JSON serialisable format

staticmethod from_data(*hsl: tuple*)
Constructor from the data received through the API or configuration

staticmethod from_hsl(*hsl: tuple*)
HSL constructor

staticmethod from_rgb(*rgb: tuple*)
RGB constructor

h
Hue

l
Lightness

rgb
RGB

s
Saturation

class homecontrol.dependencies.data_types.**DateTime**
Bases: datetime.datetime

date time format

dump()
Dump to JSON serialisable data

staticmethod from_data(*data*)
Construct from JSON serialisable data

1.4 Entity Types

Module containing the entity types Every new Item or Module will get one of these classes as a base class

class homecontrol.dependencies.entity_types.**Item**
Bases: object

A dummy Item

config_schema = <Schema(<class 'object'>, extra=PREVENT_EXTRA, required=False) object>

coroutine constructor(*identifier: str*, *name: str*, *cfg: dict*, *state_defaults: dict*, *core: homecontrol.core.Core*) → Item
Constructs an item

coroutine init() → None
Default init method

status = 'offline'

coroutine stop() → None
Default stop method

class homecontrol.dependencies.entity_types.**Module**
Bases: object

A dummy Module

folder_location = None

coroutine init() → None
Default init method

coroutine stop() → None
Default stop method

homecontrol.dependencies.entity_types.**ModuleDef**
alias of *homecontrol.dependencies.entity_types.Module*

1.5 Event Engine

EventEngine for HomeControl

class homecontrol.dependencies.event_engine.**Event**(*event_type: str*, *data: dict = None*, *timestamp: int = None*, *kwargs: dict = None*)

Bases: object

Representation for an Event

class homecontrol.dependencies.event_engine.**EventEngine**(*core*)

Bases: object

Dispatcher for events

broadcast(*event_type: str*, *data: dict = None*, ***kwargs*) → List[_asyncio.Future]

Broadcast an event and return the futures

Every listener is a coroutine that will simply receive event and *kwargs*

Example: >>> async def on_event(event: Event, ...): >>> return

broadcast_threaded(*event_type: str*, *data: dict = None*, ***kwargs*) → List[_asyncio.Task]

Same as broadcast BUT - It returns Futures and not Tasks - It uses threads

staticmethod create_event(*event_type: str*, *data: dict = None*, ***kwargs*) → homecontrol.dependencies.event_engine.Event

Creates an Event to be broadcasted

coroutine gather(*event_type: str*, *data: dict = None*, *timeout: Union[float, int, None] = None*, ***kwargs*) → List[Any]

Broadcast an event and return the results

get_event_handlers(*event: homecontrol.dependencies.event_engine.Event*) → List[T]

Returns a list of handlers for an Event

register(*event: str*) → Callable

Decorator to register event handlers

remove_handler(*event: str*, *handler: Callable*) → None

Removes an event handler

1.6 Item Manager

ItemManager for HomeControl

class homecontrol.dependencies.item_manager.**ItemManager**(*core*)

Bases: object

ItemManager manages all your stateful items

coroutine add_from_module(*mod_obj: homecontrol.dependencies.entity_types.Module*) → None

Adds the item specifications of a module to the dict of available ones

mod_obj: homecontrol.entity_types.Module

coroutine create_from_storage_entry(*storage_entry: dict*) → homecontrol.dependencies.entity_types.Item

Creates an Item from a storage entry

coroutine create_item(*identifier: str*, *item_type: str*, *cfg: dict = None*, *state_defaults: dict = None*, *name: str = None*) → homecontrol.dependencies.entity_types.Item

Creates a HomeControl item

coroutine init() → None
Initialise the items from configuration

coroutine init_item(*item: homecontrol.dependencies.entity_types.Item*) → None
Initialises an item

for ... in iter_items_by_id(*iterable*) → [<class 'homecontrol.dependencies.entity_types.Item'>]
Translates item identifiers into item instances

load_yaml_config() → None
Loads the YAML configuration

register_item(*item: homecontrol.dependencies.entity_types.Item*) → None
Registers an item

coroutine remove_item(*identifier: str*) → None
Removes a HomeControl item

identifier: str The item's identifier

coroutine stop_item(*item: homecontrol.dependencies.entity_types.Item*, *status: homecontrol.const.ItemStatus = <ItemStatus.STOPPED: 'stopped'>*) → None
Stops an item

update_storage_entry(*entry: dict*) → None
Updates a config storage entry

homecontrol.dependencies.item_manager.**yaml_entry_to_json**(*yaml_entry: dict*) → dict
Converts a yaml entry to a JSON entry

1.7 JSON Response

JSONResponse module

class homecontrol.dependencies.json_response.**JSONResponse**(*data: Any = None*, *error: str = None*, *status_code: int = 200*, *core=None*, *headers: dict = None*)
Bases: aiohttp.web_response.Response

A HTTP response for JSON data

1.8 JSON

JSON Encoder and Decoder

class homecontrol.dependencies.json.**JSONEncoder**(*core: Core*, **args*, ***kwargs*)
Bases: json.encoder.JSONEncoder

Custom JSONEncoder that also parses HomeControl types

default(*o*)
Encode custom types

homecontrol.dependencies.json.**dump**(*obj*, *fp*, ***, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *indent=None*, *separators=None*, *sort_keys=False*, *core: Core = None*, ***kw*)

Dumps an object into a Writer with support for HomeControl's data types

homecontrol.dependencies.json.**dumps**(*obj*, ***, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *indent=None*, *separators=None*, *sort_keys=False*, *core: Core = None*, ***kw*)

Dumps an object into a JSON string with support for HomeControl's data types

1.9 Module Manager

ModuleManager module

class homecontrol.dependencies.module_manager.**ModuleFolder**(*name: str*)

Bases: object

module folder representation to create a dummy package in sys.modules

class homecontrol.dependencies.module_manager.**ModuleManager**(*core*)

Bases: object

Manages your modules

coroutine init() → None

Initialise the modules

coroutine load_file_module(*mod_path: str*, *name: str*) -> (<class 'homecontrol.dependencies.entity_types.Module'>, <class 'Exception'>)

Loads a module from a file and initialises it

Returns a Module object

coroutine load_folder(*path: str*) → [<class 'object'>]

Load every module in a folder

coroutine load_folder_module(*path: str*, *name: str*) -> (<class 'homecontrol.dependencies.entity_types.Module'>, <class 'Exception'>)

Loads a module from a folder and initialises it

It also takes care of pip requirements

Returns a Module object

resource_path(*module: homecontrol.dependencies.entity_types.Module*, *path: str = ''*) → str

Returns the path for a module's resource folder Note that only folder modules can have a resource path

coroutine stop() → None

Unloads all modules to prepare for a shutdown

1.10 State Engine

StateEngine module

class homecontrol.dependencies.state_engine.**State**(*state_engine: homecontrol.dependencies.state_engine.StateEngine*, *default*, *getter: Callable = None*, *setter: Callable = None*, *name: str = None*, *state_type: type = None*, *schema: dict = None*, *poll_interval: float = None*)

Bases: object

Holds one state of an item

check_value(*value*) → voluptuous.error.Error
Checks if a value is valid for a state

coroutine get()
Gets a state

coroutine poll_value() → None
Polls the current state and updates it

coroutine set(*value*) → dict
Sets a state

coroutine update(*value*)
Updates a state

class homecontrol.dependencies.state_engine.**StateDef**(*poll_interval: Optional[float] = None*, *default: Any = None*, *default_factory: Callable = None*)

Bases: object

getter() → Callable
Decorator to register a getter

register_state(*state_engine: StateEngine*, *name: str*, *item: homecontrol.dependencies.entity_types.Item*) → State
Generates a State instance and registers it to a StateEngine

setter(*schema: Optional[voluptuous.schema_builder.Schema] = None*) → Callable
Decorator to register a setter

class homecontrol.dependencies.state_engine.**StateEngine**(*item: homecontrol.dependencies.entity_types.Item*, *core: homecontrol.core.Core*, *state_defaults: dict = None*)

Bases: object

Holds the states of an item

coroutine bulk_update(***kwargs*)
Called from an item to update multiple states

check_value(*state: str*, *value*) → voluptuous.error.Error
Checks if a value is valid for a state

coroutine dump() → dict
Return a JSON serialisable object

coroutine get(*state: str*)
Gets an item's state

register_state(*state: homecontrol.dependencies.state_engine.State*) → None
Registers a State instance to the StateEngine

coroutine set(*state: str*, *value*) → dict
Sets an item's state

coroutine update(*state: str*, *value*)
Called from an item to update its state

1.11 Throttle Function

throttle_function

class homecontrol.dependencies.throttle_function.**throttle**(*s: float = 1*)
Bases: object

Throttles a function so it only gets called at a fixed frequency

1.12 Tick Engine

TickEngine

class homecontrol.dependencies.tick_engine.**TickEngine**(*core*)
Bases: object

The TickEngine is there to handle every kind of recurring task. Using the decorator TickEngine.tick(interval) you can add a task.

remove_tick(*interval: float*, *coro*) → None
Removes a tick handler for an interval

coroutine stop()
Called on HomeControl shutdown

tick(*interval: float*) → Callable
Register a tick

interval: int Interval in seconds

1.13 Validators

1.14 YAML Loader

Provides a YAML loader

class homecontrol.dependencies.yaml_loader.**Constructor**
Bases: yaml.constructor.SafeConstructor

Constructor for yaml

env_var_constructor(*node: yaml.nodes.Node*) → str
Embeds an environment variable !env_var <name> [default]

format_string_constructor(*node: yaml.nodes.Node = None*) → str
Renders a format string .. rubric:: Example

!format { template: "Hello {who}", who: You }

include_dir_file_mapped_constructor(*node: yaml.nodes.Node = None*) → dict
!include_dir_file_mapped <folder>

Loads multiple files from a folder and maps their contents to their filenames

include_file_constructor(*node: yaml.nodes.Node = None*) → object
!include <path> ~/ for paths relative to your home directory / for absolute paths anything else for paths relative to your config folder

include_merge_constructor(*node: yaml.nodes.Node = None) -> (<class 'list'>, <class 'dict'>*)
!include <filelfolder> ...

Merges file or folder contents

This constructor only works if all the files' contents are of same type and if this type is either list or dict.

listdir_constructor(*node: yaml.nodes.Node*) → list
!listdir <path>

Returns the contents of a directory

path_constructor(*node: yaml.nodes.Node*) → str
!path <path> ~/ for paths relative to your home directory / for absolute paths anything else for paths relative to your config folder

type_constructor(*suffix: str*, *node: yaml.nodes.Node = None*) → type
Construct a custom object

vol_constructor(*suffix: str*, *node: yaml.nodes.Node = None*) → voluptuous.schema_builder.Schema
Construct a voluptuous object

class homecontrol.dependencies.yaml_loader.**YAMLLoader**(*stream*, *cfg_folder: str = None*)
Bases: yaml.reader.Reader, yaml.scanner.Scanner, yaml.parser.Parser, yaml.composer.Composer, *homecontrol.dependencies.yaml_loader.Constructor*, yaml.resolver.Resolver

Loads YAML with custom constructors

classmethod load(*data*, *cfg_folder: str = None*)
Loads data

CHAPTER 2

homecontrol module

The core instance for HomeControl

class homecontrol.core.**Core**(*cfg: dict*, *cfg_file: str*, *loop: Optional[asyncio.events.AbstractEventLoop] = None*, *start_args: Optional[Dict[KT, VT]] = None*)

Bases: object

Represents the root object for HomeControl

coroutine block_until_stop() → int
Blocking method to keep HomeControl running until Core.block_future is done

Also triggers the stop coroutine when block_future has a result

coroutine bootstrap() → None
Startup coroutine for Core

restart() → None
Restarts HomeControl

shutdown() → None
Shuts HomeControl down

coroutine stop() → None
Stops HomeControl

Constants used by HomeControl

class homecontrol.const.**ItemStatus**

Bases: enum.Enum

Every status an item can have

OFFLINE = 'offline'

ONLINE = 'online'

STOPPED = 'stopped'

WAITING_FOR_DEPENDENCY = 'waiting-for-dependency'

Exceptions

exception homecontrol.exceptions.**ConfigDomainAlreadyRegistered**
Bases: *homecontrol.exceptions.HomeControlException*

The configuration domain is already registered

exception homecontrol.exceptions.**ConfigurationNotApproved**
Bases: *homecontrol.exceptions.HomeControlException*

Configuration has not been approved

exception homecontrol.exceptions.**HomeControlException**
Bases: BaseException

The base exception for HomeControl

exception homecontrol.exceptions.**ItemNotFoundException**
Bases: *homecontrol.exceptions.HomeControlException*

Item not found

exception homecontrol.exceptions.**ItemNotOnlineError**
Bases: *homecontrol.exceptions.HomeControlException*

Item is not online

exception homecontrol.exceptions.**ItemTypeNotExistsError**
Bases: *homecontrol.exceptions.HomeControlException*

Item type does not exist

exception homecontrol.exceptions.**ModuleNotFoundException**
Bases: *homecontrol.exceptions.HomeControlException*

Module not found

exception homecontrol.exceptions.**PipInstallError**
Bases: *homecontrol.exceptions.HomeControlException*

A pip install failed

CHAPTER 3

API Module

The HTTP API for HomeControl Currenty it supports following routes:

GET /api/ping
Ping the API

Example request:

```
GET /api/ping HTTP/ HTTP/1.1
Host: homecontrol.local:8080
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

{
   "data": "PONG",
   "status_code": 200,
   "success": true
}
```

Response Headers

- Content-Type – application/json

Status Codes

- 200 OK – HomeControl is online

CHAPTER 4

Custom Scripts

This module offers a very simple way to execute custom code in HomeControl Please note that this method is quite dangerous as these scripts could do anything and if you're running HomeControl as root they will even have root privileges

CHAPTER 5

PiGPIO Adapter

Provides an interface with a Raspberry Pi's GPIO ports using pigpio.

5.1 Installation

You can find an installation guide on the official website.

On Raspbian you only need to run

```
sudo apt update
sudo apt install pigpio
```

The simplest way to automatically run pigpio on boot would be:

```
sudo bash -c "echo /usr/bin/pigpiod >> /etc/rc.local"
```

Python Module Index

h

HTTP Routing Table

/api

Index

F

G

H

I

J

L

M

O

P

R

S

T

U

V

W

Y