Hein Utilities

Release 3.14.2

Jul 05, 2021

Contents:

Hein Utilities is a collection of utility classes and methods which are used in a variety of Hein Group projects.

CHAPTER 1

hein_utilities package

1.1 hein_utilities.slack_integration package

1.1.1 hein_utilities.slack_integration.bots module

1.1.2 hein_utilities.slack_integration.logging module

1.1.3 hein_utilities.slack_integration.parsing module

1.1.4 hein_utilities.slack_integration.slack_managers module

1.2 hein_utilities.control_manager module

Contextual action managers

class hein_utilities.control_manager.**ControlManager**(*start_action: Callable = <function do_nothing>, stop_action: Callable = <function do_nothing>, resume_action: Callable = <function do_nothing>, pause_action: Callable = <function do_nothing>, status_query: Callable = <function return_empty>*)

Bases: `object`

A control manager for executing steps contextually. Context actions can be assigned as properties of the manager so that they can be executed by the appropriate trigger.

Parameters

- **start_action** – callable action to execute on a start trigger

- **stop_action** – callable action to execute on a stop trigger
- **resume_action** – callable action to execute on a resume trigger
- **pause_action** – callable action to execute on a pause trigger
- **status_query** – callable which returns a string representing the status of the thing being controlled

pause_action
the callable which is executed on pause catches

pause_pattern = re.compile('pause|suspend', re.IGNORECASE)

resume_action
the callable which is executed on resume catches

resume_pattern = re.compile('resume', re.IGNORECASE)

start_action
the callable which is executed on start catches

start_pattern = re.compile('start', re.IGNORECASE)

status_pattern = re.compile('status', re.IGNORECASE)

status_query
the callable which is used to retrieve a status string

stop_action
the callable which is executed on stop catches

stop_pattern = re.compile('stop|halt|quit', re.IGNORECASE)

hein_utilities.control_manager.**do_nothing**(**args*, ***kwargs*)
function that does nothing

hein_utilities.control_manager.**return_empty**(**args*, ***kwargs*)
function that returns an empty string

1.3 hein_utilities.datetime_utilities module

utility class for parsing and manipulating datetime objects

hein_utilities.datetime_utilities.**array_of_datetime_objects_to_array_of_relative_datetime_**

Legacy passthrough for deprecated function. Use the datetimeManager class method instead.

hein_utilities.datetime_utilities.**array_of_str_to_array_of_datetime_objects**(*list_of_string_values*, *date-time_format: str = '%Y_%m_%d_%H_%* → List[datetime.datetim

Legacy passthrough for deprecated function. Use the datetimeManager class method instead.

class hein_utilities.datetime_utilities.**datetimeManager**(*string_format: str = '%Y_%m_%d_%H_%M_%S'*)

Bases: object

utility class for parsing and manipulating datetime objects

Parameters **string_format** – datetime string format to use for parsing

now_string() → str

Get the current time from datetime.now() formatted as as a string according to the string_format property :return:

classmethod relative_datetime(*datetime_objects: List[datetime.datetime], units: Union[seconds, minutes, hours] = None, rounding: int = None*) → List[datetime.timedelta]

Convert an array of datetime objects that are absolute times, and return an array where all the times in the array are relative to the first time in the array. The relativity can be in seconds, minutes, or hours.

Parameters

- **datetime_objects** – a list of datetime objects
- **units** – one of _unit_seconds, _unit_minutes, or _unit_hours
- **rounding** (*int,*) – the number of decimal places to round to

Returns

classmethod str_to_datetime(**string_values, datetime_format: str = None*) → List[datetime.datetime]

Convert a list of string values, back into datetime objects with a specific format; in this case, the string has to have been a datetime object that was converted into a string with the datetime_format that is passed in this function.

Main use has previously been when files where timestamped, and when the file names need to be converted back into datetime objects in order to do calculations

Parameters

- **string_values** – one or more (a list) of strings that can be converted into datetime objects
- **datetime_format** (*str,*) – a string to represent the datetime format the string should be converted into; it should also have been the format that the strings already are in

Returns

string_format

1.4 hein_utilities.files module

class hein_utilities.files.**Watcher**(*path: Union[str, pathlib.Path], watchfor: Union[str, Pattern[AnyStr]] = '', includesubfolders: bool = True, subdirectory: str = None, watch_type: str = 'file', exclude_subfolders: List[str] = None, match_exact: bool = False, ignore_case: bool = True*)

Bases: object

Watches a folder for file changes.

Parameters

- **path** – The folder path to watch for changes

- **watchfor** – Watch for this item. This can be a full filename, or an extension (denoted by ., *e.g.* ".ext")
- **includesubfolders** (*bool*) – wehther to search subfolders
- **subdirectory** (*str*) – specified subdirectory
- **watch_type** (*str*) – The type of item to watch for ('file' or 'folder')
- **exclude_subfolders** – specific sub folders to exclude when looking for files. These subfolders will be globally excluded.
- **match_exact** – whether the watcher should only look for exact matches of the provided pattern
- **ignore_case** – case sensitivity for matches

check_path_for_files()
Finds all instances of the watchfor item in the path

contents
Finds all instances of the watchfor item in the path

find_subfolder()
returns the subdirectory path within the full path where the target file is

full_path
full watch path (including subdirectory)

ignore_case
whether instance will be case sensitive

newest_instance()
Retrieves the newest instance of the watched files.

Returns path to newest instance

Return type str

oldest_instance(*wait=False*, ***kwargs*)
Retrieves the first instance of the watched files.

Parameters **wait** – if there are no instances, whether to wait for one to appear

Returns path to first instance (None if there are no files present)

path
path to watch

subdirectory
specific subdirectory to watch within the path

wait_for_presence(*waittime=1.0*) → List[str]
waits for a specified match to appear in the watched path

watchfor
the pattern to watch for in the directory

1.5 hein_utilities.json_utilities module

json interaction methods

hein_utilities.json_utilities.**create_json_file**(*json_file_path: str*)

Parameters **json_file_path** (*str,*) – path to a json file

Returns

hein_utilities.json_utilities.**load_json_file**(*json_file_path*)

hein_utilities.json_utilities.**update_json_file**(*json_file_path: str, data_to_update_with: dict*)

update a json file with data. the data must be in the form of a dictionary.

if the key of the dictionary value to add into the json file doesn't exist in the json file, create a new key value pair based on the data passed through.

if the key of the dictionary value to add into the json file does exist in the json file and the value is a dictionary itself, update the current value to include the new dictionary value.

if the key of the dictionary value to add into the json file does exist in the json file and the value is a list itself, update the current value to include the new list value.

if the key of the dictionary value to add into the json file does exist in the json file and the value is a single value that isn't a dictionary or list, overwrite the value

Parameters

- **json_file_path** –
- **data_to_update_with** –

Returns

1.6 hein_utilities.misc module

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

h

Index

U

W