vartrix

Jan 26, 2023

Contents:

CHAPTER 1

vartrix package

The *vartrix* package enables powerful control over parameters used across a package for analyses. It can be very useful when hundreds of parameters are required that are particular to unique analyses runs. Or, when studies over combinations of ranges of parameters are needed.

Check out the README for tutorials to see it in action.

1.1 Module contents

Created on Mon Aug 26 16:12:03 2019

@author: Reuben

1.2 Submodules

1.3 vartrix.automate module

Created on Mon Aug 26 22:12:43 2019

@author: Reuben

class vartrix.automate.**Automation_Set**(*name*, *sequences=None*)

Bases: object

add(*sequence*)

all_sequences()

build(*data*)

clear()

run(*container*, *obj*, *seq_name=None*, *aliases=None*)

Run an automation set

Parameters

- **set_name** (*str*) – The name of the set to run
- **obj** (*object*) – The automated class instance.
- **seq_name** (*str*) – [Optional] A specific sequence within the set to run exclusively.

sequence_names

sequences

set_sequences(*sequences*)

class vartrix.automate.**Automator**(*container*, *fname=None*, *data=None*, *aliases=None*)
Bases: object

build(*data*)

run(*set_name*, *obj*, *seq_name=None*)

set_aliases(*aliases*)

set_automation_set(*set_name*, *automation_set*)

class vartrix.automate.**Constant**(*name*, *data=None*)
Bases: *vartrix.automate.Vector*

setup(*data*)

class vartrix.automate.**Csv_File**(*name*, *data=None*)
Bases: *vartrix.automate.Vector*

setup(*data*)

class vartrix.automate.**Method**(*name*, *vectors=None*)
Bases: object

add(*vector*)

build(*data*, *vectors*)

execute(*container*, *obj*, *aliases=None*, *info=None*)

get_lst(*vectors=None*, *typ='values'*)

name

set_vectors(*vectors*)

show(*info*, *label_dct*, *complete=None*)

class vartrix.automate.**Sequence**(*name*, *methods=None*)
Bases: object

add(*method*)

build(*data*, *vectors*)

execute(*container*, *obj*, *aliases=None*, *info=None*)

methods

name

set_methods(*methods=None*)

class vartrix.automate.**Value_Dictionaries**(*name*, *data=None*)
Bases: *vartrix.automate.Vector*

setup(*data*)

class vartrix.automate.**Value_Lists**(*name*, *data=None*)
Bases: *vartrix.automate.Vector*

setup(*data*)

transpose_dict(*dct*)

class vartrix.automate.**Vector**(*name*, *data=None*)
Bases: object

Subclass for different entry formats

get_label_lst()

initialise(*data*)

setup(*data*)

values(*typ='values'*)

class vartrix.automate.**Vectors**
Bases: object

build(*data*)

build_one(*data*, *name*)

clear()

get_vector(*key*)

set_style(*key*, *style*)

set_vector(*key*, *vector*)

vartrix.automate.**safe_call**(*method_name*, *obj*, **args*, ***kwargs*)

vartrix.automate.**set_root**(*new*)
This shouldn't be done

1.4 vartrix.container module

Created on Tue Aug 27 19:52:22 2019

@author: Reuben

This module includes the Container class upon which all of vartrix is built.

The idea of a container is to contain all of the key-value pairs for some set of variables or parameters. Keys have an implied heirarchy using a dot-format. For example, 'A.b' means that 'b' is a subkey of 'A'. All keys are stored in a simple flat dictionary format, but some extra methods allow the heirarchy to be used to simplify usage.

class vartrix.container.**Container**(*dct=None*)
Bases: dict

A dictionary-like class that updates observers

Parameters **dct** (*dict*) – A dictionary (possibly nested) of key-value pairs to use.

Note: A *dotkey* is a dictionary key in the Container. It's called a dotkey simply because it's designed to have dots in it to represent a heirarchy.

---

add(*dct*)
Add another set of data to the container

context(*dct*, *safe=True*)
A context manager for temporary changes in values

Parameters

- **dct** (*dict*) – A dictionary of dotkey-value pairs.
- **safe** (*bool*) – [Optional] set to false to ignore locks. Keys must already exist in the container.

copy() → a shallow copy of D

dset(*dct*, *safe=False*, *update_backup=False*)
Set multiple values specified in a dictionary

Parameters

- **dct** (*dict*) – The dictionary of key-value pairs.
- **safe** (*bool*) – Optional boolean. If true, the key must already exist in the Container instance.
- **update_backup** (*bool*) – If true, update the internal backup values that reset() restores.

load(*dct*)
Set the container data using a dictionary

lock(*key*)

classmethod merge(*containers*)
Combine a list of containers

reset()

set(*key*, *val*, *safe=False*)
Set the value of a key

Parameters

- **key** (*str*) – The key value
- **val** – The value to set. It can be a numpy array.
- **safe** (*bool*) – Optional boolean. If true, the dotkey must already
- **in the Container instance.** (*exist*) –

set_aliases(*aliases*)

to_dict()

unlock(*key*)

1.5 vartrix.namespace module

Created on Thu Sep 5 11:14:59 2019

@author: Reuben

class vartrix.namespace.**Name_Space**(*obj_cls=<class 'vartrix.container.Container'>*)
Bases: dict

active_container()

create(*key*, *dct=None*)

duplicate(*key*, *new_key*)

get(*key=None*, *dct=None*)
Return the value for key if key is in the dictionary, else default.

set_active(*key*)

vartrix.namespace.**get_container**(*name=None*)

1.6 vartrix.persist module

Created on Mon Sep 9 17:05:43 2019

@author: Reuben

class vartrix.persist.**Handler**
Bases: object

suitable(*fname*, ***kwargs*)

class vartrix.persist.**Manager**
Bases: object

add_handler(*key*, *handler*)

default_handler = 'yaml'

load(*source*, *handler=None*, ***kwargs*)

save(*dct*, *target*, *handler=None*, ***kwargs*)

specify(*key*)

class vartrix.persist.**Xlsx**
Bases: *vartrix.persist.Handler*

load(*fname*, *keys='dotkeys'*, *values='values'*, *aliases=None*, ***kwargs*)

save(*dct*, *fname*, *loadname*, *keys='dotkeys'*, *values='values'*, *aliases=None*, ***kwargs*)

class vartrix.persist.**Yaml**
Bases: *vartrix.persist.Handler*

load(*fname*, ***kwargs*)

save(*dct*, *fname*, *flow_style=False*, ***kwargs*)

vartrix.persist.**is_importable**(*module_name*)

1.7 vartrix.settings module

Created on Mon Aug 26 16:01:18 2019

@author: Reuben

1.8 vartrix.utils module

Created on Sat Sep 14 11:36:50 2019

@author: Reuben

This module contains helper classes and functions.

class vartrix.utils.**Attrdict**
Bases: dict

class vartrix.utils.**Factory**(*name*, *container*, *dotkey*, *build_function=None*, *class_method=None*, *default=None*)
Bases: object

Instantiate module classes based on container key

Parameters

- **name** (*str*) – The module name (often *__name__*)
- **container** (Container) – The container instance
- **dotkey** (*str*) – The dotkey of the class name.
- **build_function** (*func*) – OPTIONAL: A function that takes the new
- **object. If provided, the return value of the Factory is** (*instance*) –
- **return value of the function.** (*the*) –
- **class_method** (*str*) – OPTIONAL: The name of a class method called
- **instantiate the object.** (*to*) –
- **default** (*str*) – OPTIONAL: An optional default value if the dotkey
- **not present.** (*is*) –

Note: The class name pointed to by the dotkey must exist in the module, or a KeyError will be raised.

new(**args*, ***kwargs*)
Create a new instance based on the current dotkey value

class vartrix.utils.**Simple_Factory**(*name*, *build_function=None*, *class_method=None*)
Bases: object

new(*cls_name*, **args*, ***kwargs*)

vartrix.utils.**denumpify**(*obj*)

vartrix.utils.**flat**(*dct*, *base=''*)
Create a flat dictionary representation of nested dictionary

vartrix.utils.**nested**(*dct*)
Create a nested dictionary representation of a dotkey flat dictionary

vartrix.utils.**numpify**(*obj*)

1.9 vartrix.view module

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

v

Index

R

S

T

U

V

X

Y