Watson - Common

Release 1.4.2

July 29, 2016

Contents

1 Build Status **3**

2 Installation **5**

3 Testing **7**

4 Contributing **9**

5 Table of Contents **11**

5.1 Reference Library 11

Python Module Index **15**

Useful common utility functions and classes for working with lists, dictionaries, importlib and xml.

CHAPTER 1

Build Status

CHAPTER 2

Installation

```
pip install watson-common
```

CHAPTER 3

Testing

Watson can be tested with py.test. Simply activate your virtualenv and run `python setup.py test`.

CHAPTER 4

Contributing

If you would like to contribute to Watson, please feel free to issue a pull request via Github with the associated tests for your code. Your name will be added to the AUTHORS file under contributors.

CHAPTER 5

Table of Contents

5.1 Reference Library

5.1.1 watson.common.contextmanagers

5.1.2 watson.common.datastructures

class `watson.common.datastructures.`**`ImmutableDict`**(**args*)

Creates an immutable dict.

While not truly immutable (_mutable can be changed), it works effectively in the same fashion.

`__init__`(**args*)

class `watson.common.datastructures.`**`ImmutableMultiDict`**(**args*)

Creates an immuatable MultiDict.

`__init__`(**args*)

class `watson.common.datastructures.`**`MultiDict`**(*args=None*)

A dictionary type that can contain multiple items for a single key.

Dictionary type that will create a list of values if more than one item is set for that particular key.

Example:

```
multi_dict = MultiDict()
multi_dict['one'] = 1
multi_dict['one'] = 'itchi'
print(multi_dict)  # {'one': [1, 'itchi']}
```

`__init__`(*args=None*)

`set`(*key*, *value*, *replace=False*)

Add a new item to the dictionary.

Set the key to value on the dictionary, converting the existing value to a list if it is a string, otherwise append the value.

Parameters

- **`key`** (*`string`*) – The key used to the store the value.
- **`value`** (*`mixed`*) – The value to store.
- **`replace`** (*`bool`*) – Whether or not the value should be replaced.

Example:

```
multi_dict = MultiDict()
multi_dict.set('item', 'value')  # or multi_dict['item'] = 'value'
```

watson.common.datastructures.**dict_deep_update**(*d1*, *d2*)

Recursively merge two dictionaries.

Merges two dictionaries together rather than a shallow update().

Parameters

- **d1** (*dict*) – The original dict.
- **d2** (*dict*) – The dict to merge with d1.

Returns A new dict containing the merged dicts.

Return type dict

watson.common.datastructures.**merge_dicts**(**dicts*)

Merges multiple dictionaries and returns a single new dict.

Unlike dict.update this will create a new dict.

Parameters **dicts** (*list*) – The dicts that are being merged

Returns A new dict containing the merged dicts

Return type dict

watson.common.datastructures.**module_to_dict**(*module*, *ignore_starts_with='*')

Load the contents of a module into a dict.

Parameters **ignore_starts_with** (*string*) – Ignore all module keys that begin with this value.

Returns The contents of the module as a dict

Return type dict

Example:

```
# my_module.py contents:
# variable = 'value'
import my_module
a_dict = module_to_dict(my_module)
a_dict['variable']
```

5.1.3 watson.common.decorators

class watson.common.decorators.**cached_property**(*func*)

Allows expensive property calls to be cached.

Once the property is called, it's result is stored in the corresponding property name prefixed with an underscore.

Example:

```
class MyClass(object):
    @cached_property
    def expensive_call(self):
        # do something expensive

klass = MyClass()
```

```
klass.expensive_call  # initial call is made
klass.expensive_call  # return value is retrieved from an internal cache
del klass._expensive_call
```

__init__(*func*)

watson.common.decorators.**instance_set**(*ignore=None*)

Allows initial binding of arguments to an __init__ method.

Example:

```
class MyClass(object):
    value = None

    @instance_set
    def __init__(self, value="Instance variable declaration"):
        pass

klass = MyClass()
klass.value  # Bound Value
```

5.1.4 watson.common.imports

watson.common.imports.**get_qualified_name**(*obj*)

Retrieve the full module path of an object.

Example:

```
from watson.http.messages import Request
request = Request()
name = get_qualified_name(request) # watson.http.messages.Request
```

watson.common.imports.**load_definition_from_string**(*qualified_module*, *cache=True*)

Load a definition based on a fully qualified string.

Returns None or the loaded object

Example:

```
definition = load_definition_from_string('watson.http.messages.Request')
request = definition()
```

W

watson.common.contextmanagers, 11
watson.common.datastructures, 11
watson.common.decorators, 12
watson.common.imports, 13

Symbols

__init__() (watson.common.datastructures.ImmutableDict method), 11
__init__() (watson.common.datastructures.ImmutableMultiDict method), 11
__init__() (watson.common.datastructures.MultiDict method), 11
__init__() (watson.common.decorators.cached_property method), 13

C

cached_property (class in watson.common.decorators), 12

D

dict_deep_update() (in module watson.common.datastructures), 12

G

get_qualified_name() (in module watson.common.imports), 13

I

ImmutableDict (class in watson.common.datastructures), 11
ImmutableMultiDict (class in watson.common.datastructures), 11
instance_set() (in module watson.common.decorators), 13

L

load_definition_from_string() (in module watson.common.imports), 13

M

merge_dicts() (in module watson.common.datastructures), 12
module_to_dict() (in module watson.common.datastructures), 12
MultiDict (class in watson.common.datastructures), 11

S

set() (watson.common.datastructures.MultiDict method), 11

W

watson.common.contextmanagers (module), 11
watson.common.datastructures (module), 11
watson.common.decorators (module), 12
watson.common.imports (module), 13