# Privex Python Helpers Documentation

**Privex Inc., Chris (Someguy123)**

**Aug 23, 2019**

# MAIN:

Welcome to the documentation for Privex's Python Helpers - a small, open source Python 3 package containing a variety of functions, classes, exceptions, decorators and more - each of which would otherwise be too small to maintain in an individual package.

This documentation is automatically kept up to date by ReadTheDocs, as it is automatically re-built each time a new commit is pushed to the Github Project

CHAPTER

ONE

# CONTENTS

## 1.1 Installation

### 1.1.1 Download and install from PyPi using pip (recommended)

```
pip3 install privex-helpers
```

### 1.1.2 (Alternative) Manual install from Git

**Option 1 - Use pip to install straight from Github**

```
pip3 install git+https://github.com/Privex/python-helpers
```

**Option 2 - Clone and install manually**

```
# Clone the repository from Github
git clone https://github.com/Privex/python-helpers
cd python-helpers

# RECOMMENDED MANUAL INSTALL METHOD
# Use pip to install the source code
pip3 install .

# ALTERNATIVE MANUAL INSTALL METHOD
# If you don't have pip, or have issues with installing using it, then you can use␣
↪setuptools instead.
python3 setup.py install
```

## 1.2 Code Docs for each helper module

| | |
|---|---|
| *privex.helpers.common* | Common functions and classes that don't fit into a specific category |
| *privex.helpers.decorators* | Class Method / Function decorators |
| *privex.helpers.django* | This module file contains Django-specific helper functions, to help save time when developing with the Django framework. |

Continued on next page

Table 1 – continued from previous page

| | |
|---|---|
| *privex.helpers.exceptions* | Exception classes used either by our helpers, or just generic exception names which are missing from the standard base exceptions in Python, and are commonly used across our projects. |
| *privex.helpers.net* | Network related helper code |
| *privex.helpers.plugin* | This module handles connection objects for databases, APIs etc. |
| *privex.helpers.settings* | Configuration options for helpers, and services they depend on, such as Redis. |

### 1.2.1 privex.helpers.common

Common functions and classes that don't fit into a specific category

**Copyright**:

```
    +==================================================+
    |                 © 2019 Privex Inc.               |
    |               https://www.privex.io              |
    +==================================================+
    |                                                  |
    |        Originally Developed by Privex Inc.       |
    |                                                  |
    |        Core Developer(s):                        |
    |                                                  |
    |          (+)  Chris (@someguy123) [Privex]       |
    |          (+)  Kale (@kryogenic) [Privex]         |
    |                                                  |
    +==================================================+

Copyright 2019     Privex Inc.   ( https://www.privex.io )

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to use,
copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the
Software, and to permit persons to whom the Software is furnished to do so,
subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

#### Functions

| | |
|---|---|
| *empty*(v[, zero, itr]) | Quickly check if a variable is empty or not. |

Continued on next page

Table 2 – continued from previous page

| | |
|---|---|
| *env_csv*(env_key[, env_default, csvsplit]) | Quick n' dirty parsing of simple CSV formatted environment variables, with fallback to user specified `env_default` (defaults to None) |
| *env_keyval*(env_key[, env_default, valsplit, ...]) | Parses an environment variable containing `key:val, key:val` into a list of tuples [(key,val), (key,val)] |
| *env_bool*(env_key[, env_default]) | Obtains an environment variable `env_key`, if it's empty or not set, `env_default` will be returned. |
| *is_false*(v[, chk_none]) | **Warning:** Unless you specifically need to verify a value is Falsey, it's usually safer to check for truth *is_true()* and invert the result, i.e. |
| *is_true*(v) | Check if a given bool/str/int value is some form of `True`: |
| *parse_csv*(line[, csvsplit]) | Quick n' dirty parsing of a simple comma separated line, with automatic whitespace stripping of both the `line` itself, and the values within the commas. |
| *parse_keyval*(line[, valsplit, csvsplit]) | Parses a csv with key:value pairs such as. |
| *random_str*([size, chars]) | Generate a random string of arbitrary length using a given character set (string / list / tuple). |

#### 1.2.1.1 privex.helpers.common.empty

privex.helpers.common.**empty**(*v*, *zero: bool = False*, *itr: bool = False*) → bool

Quickly check if a variable is empty or not. By default only '' and None are checked, use `itr` and `zero` to test for empty iterable's and zeroed variables.

Returns `True` if a variable is `None` or `''`, returns `False` if variable passes the tests

Example usage:

```
>>> x, y = [], None
>>> if empty(y):
...     print('Var y is None or a blank string')
...
>>> if empty(x, itr=True):
...     print('Var x is None, blank string, or an empty dict/list/iterable')
```

**Parameters**

- **v** – The variable to check if it's empty
- **zero** – if `zero=True`, then return `True` if the variable is int `0` or str `'0'`
- **itr** – if `itr=True`, then return `True` if the variable is `[]`, `{}`, or is an iterable and has 0 length

**Return bool is_blank** `True` if a variable is blank (`None`, `''`, `0`, `[]` etc.)

**Return bool is_blank** `False` if a variable has content (or couldn't be checked properly)

#### 1.2.1.2 privex.helpers.common.env_csv

privex.helpers.common.**env_csv**(*env_key: str*, *env_default=None*, *csvsplit=','*) → List[str]

Quick n' dirty parsing of simple CSV formatted environment variables, with fallback to user specified `env_default` (defaults to None)

Example:

```
>>> os.setenv('EXAMPLE', '  hello ,  world, test')
>>> env_csv('EXAMPLE', [])
['hello', 'world', 'test']
>>> env_csv('NONEXISTANT', [])
[]
```

**Parameters**

- **env_key** (*str*) – Environment var to attempt to load
- **env_default** (*any*) – Fallback value if the env var is empty / not set (Default: None)
- **csvsplit** (*str*) – A character (or several) used to terminate each value in the list. Default: comma `,`

**Return List[str] parsed_data** A list of str values parsed from the env var

#### 1.2.1.3 privex.helpers.common.env_keyval

privex.helpers.common.**env_keyval**(*env_key: str*, *env_default=None*, *valsplit=':'*, *csvsplit=','*) → List[Tuple[str, str]]

Parses an environment variable containing `key:val,key:val` into a list of tuples [(key,val), (key,val)]

See *parse_keyval()*

**Parameters**

- **env_key** (*str*) – Environment var to attempt to load
- **env_default** (*any*) – Fallback value if the env var is empty / not set (Default: None)
- **valsplit** (*str*) – A character (or several) used to split the key from the value (default: colon `:`)
- **csvsplit** (*str*) – A character (or several) used to terminate each keyval pair (default: comma `,`)

#### 1.2.1.4 privex.helpers.common.env_bool

privex.helpers.common.**env_bool**(*env_key: str*, *env_default=None*) → Optional[bool]

Obtains an environment variable `env_key`, if it's empty or not set, `env_default` will be returned. Otherwise, it will be converted into a boolean using *is_true()*

Example:

```
>>> os.environ['HELLO_WORLD'] = '1'
>>> env_bool('HELLO_WORLD')
True
>>> env_bool('HELLO_NOEXIST')
None
>>> env_bool('HELLO_NOEXIST', 'error')
'error'
```

**Parameters**

- **env_key** (*str*) – Environment var to attempt to load
- **env_default** (*any*) – Fallback value if the env var is empty / not set (Default: None)

#### 1.2.1.5 privex.helpers.common.is_false

privex.helpers.common.**is_false**(*v*, *chk_none: bool = True*) → bool

**Warning:** Unless you specifically need to verify a value is Falsey, it's usually safer to check for truth *is_true()* and invert the result, i.e. `if not is_true(v)`

Check if a given bool/str/int value is some form of `False`:

- **bool**: `False`
- **str**: `'false'`, `'no'`, `'n'`, `'0'`
- **int**: `0`

If `chk_none` is True (default), will also consider the below values to be Falsey:

```
boolean: None // string: 'null', 'none', ''
```

(note: strings are automatically .lower()'d)

Usage:

```
>>> is_false(0)
True
>>> is_false('yes')
False
```

**Parameters**

- **v** (*Any*) – The value to check for falseyness
- **chk_none** (*bool*) – If `True`, treat `None`/`'none'`/`'null'` as Falsey (default `True`)

**Return bool is_False** `True` if the value appears to be falsey, otherwise `False`.

#### 1.2.1.6 privex.helpers.common.is_true

privex.helpers.common.**is_true**(*v*) → bool

Check if a given bool/str/int value is some form of `True`:

- **bool**: `True`
- **str**: `'true'`, `'yes'`, `'y'`, `'1'`
- **int**: `1`

(note: strings are automatically .lower()'d)

Usage:

```
>>> is_true('true')
True
>>> is_true('no')
False
```

**Parameters** **v** (*Any*) – The value to check for truthfulness

**Return bool is_true** `True` if the value appears to be truthy, otherwise `False`.

#### 1.2.1.7 privex.helpers.common.parse_csv

privex.helpers.common.**parse_csv**(*line: str*, *csvsplit: str = ','*) → List[str]

Quick n' dirty parsing of a simple comma separated line, with automatic whitespace stripping of both the `line` itself, and the values within the commas.

Example:

```
>>> parse_csv('  hello ,  world, test')
['hello', 'world', 'test']
>>> parse_csv(' world  ;   test   ; example', csvsplit=';')
['world', 'test', 'example']
```

**Parameters** **csvsplit** (*str*) – A character (or several) used to terminate each value in the list. Default: comma `,`

#### 1.2.1.8 privex.helpers.common.parse_keyval

privex.helpers.common.**parse_keyval**(*line: str*, *valsplit: str = ':'*, *csvsplit=','*) → List[Tuple[str, str]]

Parses a csv with key:value pairs such as:

```
John Alex:Doe,Jane Sarah:Doe
```

Into a list with tuple pairs (can be easily converted to a dict):

```
[
    ('John Alex', 'Doe'),
    ('Jane Sarah', 'Doe')
]
```

By default, uses a colons `:` to split the key/value, and commas `,` to terminate the end of each keyval pair. This can be overridden by changing valsplit/csvsplit.

**Parameters**

- **line** (*str*) – A string of key:value pairs separated by commas e.g. `John Alex:Doe,Jane Sarah:Doe`
- **valsplit** (*str*) – A character (or several) used to split the key from the value (default: colon `:`)
- **csvsplit** (*str*) – A character (or several) used to terminate each keyval pair (default: comma `,`)

**Return List[Tuple[str,str]] parsed_data** A list of (key, value) tuples that can easily be casted to a dict()

#### 1.2.1.9 privex.helpers.common.random_str

privex.helpers.common.**random_str**(*size: int = 50*, *chars: Sequence = 'abcdefhkmnprstwxyz23456789ACDEFGHJKLMNPRSTWXYZ'*) → str

Generate a random string of arbitrary length using a given character set (string / list / tuple). Uses Python's SystemRandom class to provide relatively secure randomness from the OS. (On Linux, uses /dev/urandom)

By default, uses the character set `SAFE_CHARS` which contains letters a-z / A-Z and numbers 2-9 with commonly misread characters removed (such as `1`, `l`, `L`, `0` and `o`). Pass `ALPHANUM` as *chars* if you need the full set of upper/lowercase + numbers.

Usage:

```
>>> from privex.helpers import random_str
>>> # Default random string - 50 character alphanum without easily mistaken chars
>>> password = random_str()
'MrCWLYMYtT9A7bHc5ZNE4hn7PxHPmsWaT9GpfCkmZASK7ApN8r'
>>> # Customised random string - 12 characters using only the characters␣
↪`abcdef12345`
>>> custom = random_str(12, chars='abcdef12345')
'aba4cc14a43d'
```

Warning: As this relies on the OS's entropy features, it may not be cryptographically secure on non-Linux platforms:

> The returned data should be unpredictable enough for cryptographic applications, though its exact quality > depends on the OS implementation.

**Parameters**

- **size** (*int*) – Length of random string to generate (default 50 characters)
- **chars** (*str*) – Characterset to generate with ( default is `SAFE_CHARS` - a-z/A-Z/0-9 with often misread chars removed)

## Classes

| | |
|---|---|
| *ErrHelpParser*([prog, usage, description, ...]) | ErrHelpParser - Use this instead of `argparse.ArgumentParser` to automatically get full help output as well as the error message when arguments are invalid, instead of just an error message. |

privex.helpers.common.**ALPHANUM = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnopqrstu**

All characters from a-z, A-Z, and 0-9 - for random strings where there's no risk of user font confusion

**class** privex.helpers.common.**ErrHelpParser**(*prog=None*, *usage=None*, *description=None*, *epilog=None*, *parents=[]*, *formatter_class=<class 'argparse.HelpFormatter'>*, *prefix_chars='-'*, *fromfile_prefix_chars=None*, *argument_default=None*, *conflict_handler='error'*, *add_help=True*, *allow_abbrev=True*)

ErrHelpParser - Use this instead of `argparse.ArgumentParser` to automatically get full help output as well as the error message when arguments are invalid, instead of just an error message.

```
>>> parser = ErrHelpParser(description='My command line app')
>>> parser.add_argument('nums', metavar='N', type=int, nargs='+')
```

**error**(*message: string*)

Prints a usage message incorporating the message to stderr and exits.

If you override this in a subclass, it should not return – it should either exit or raise an exception.

privex.helpers.common.**SAFE_CHARS = 'abcdefhkmnprstwxyz23456789ACDEFGHJKLMNPRSTWXYZ'**

Characters that shouldn't be mistaken, avoiding users confusing an o with a 0 or an l with a 1 or I

privex.helpers.common.**empty**(*v*, *zero: bool = False*, *itr: bool = False*) → bool

Quickly check if a variable is empty or not. By default only '' and None are checked, use `itr` and `zero` to test for empty iterable's and zeroed variables.

Returns `True` if a variable is `None` or `''`, returns `False` if variable passes the tests

Example usage:

```
>>> x, y = [], None
>>> if empty(y):
...     print('Var y is None or a blank string')
...
>>> if empty(x, itr=True):
...     print('Var x is None, blank string, or an empty dict/list/iterable')
```

**Parameters**

- **v** – The variable to check if it's empty
- **zero** – if `zero=True`, then return `True` if the variable is int `0` or str `'0'`
- **itr** – if `itr=True`, then return `True` if the variable is `[]`, `{}`, or is an iterable and has 0 length

**Return bool is_blank** `True` if a variable is blank (`None`, `''`, `0`, `[]` etc.)

**Return bool is_blank** `False` if a variable has content (or couldn't be checked properly)

privex.helpers.common.**env_bool**(*env_key: str*, *env_default=None*) → Optional[bool]

Obtains an environment variable `env_key`, if it's empty or not set, `env_default` will be returned. Otherwise, it will be converted into a boolean using *`is_true()`*

Example:

```
>>> os.environ['HELLO_WORLD'] = '1'
>>> env_bool('HELLO_WORLD')
True
>>> env_bool('HELLO_NOEXIST')
None
>>> env_bool('HELLO_NOEXIST', 'error')
'error'
```

**Parameters**

- **env_key** (*str*) – Environment var to attempt to load
- **env_default** (*any*) – Fallback value if the env var is empty / not set (Default: None)

privex.helpers.common.**env_csv**(*env_key: str*, *env_default=None*, *csvsplit=','*) → List[str]

Quick n' dirty parsing of simple CSV formatted environment variables, with fallback to user specified `env_default` (defaults to None)

Example:

```
>>> os.setenv('EXAMPLE', '  hello ,  world, test')
>>> env_csv('EXAMPLE', [])
['hello', 'world', 'test']
>>> env_csv('NONEXISTANT', [])
[]
```

**Parameters**

- **env_key** (*str*) – Environment var to attempt to load
- **env_default** (*any*) – Fallback value if the env var is empty / not set (Default: None)
- **csvsplit** (*str*) – A character (or several) used to terminate each value in the list. Default: comma `,`

**Return List[str] parsed_data** A list of str values parsed from the env var

`privex.helpers.common.`**`env_keyval`**(*env_key: str*, *env_default=None*, *valsplit=':'*, *csvsplit=','*) → List[Tuple[str, str]]

Parses an environment variable containing `key:val,key:val` into a list of tuples [(key,val), (key,val)]

See *parse_keyval()*

**Parameters**

- **env_key** (*str*) – Environment var to attempt to load
- **env_default** (*any*) – Fallback value if the env var is empty / not set (Default: None)
- **valsplit** (*str*) – A character (or several) used to split the key from the value (default: colon `:`)
- **csvsplit** (*str*) – A character (or several) used to terminate each keyval pair (default: comma `,`)

`privex.helpers.common.`**`is_false`**(*v*, *chk_none: bool = True*) → bool

**Warning:** Unless you specifically need to verify a value is Falsey, it's usually safer to check for truth *is_true()* and invert the result, i.e. `if not is_true(v)`

Check if a given bool/str/int value is some form of `False`:

- **bool**: `False`
- **str**: `'false'`, `'no'`, `'n'`, `'0'`
- **int**: `0`

If `chk_none` is True (default), will also consider the below values to be Falsey:

```
boolean: None // string: 'null', 'none', ''
```

(note: strings are automatically .lower()'d)

Usage:

```
>>> is_false(0)
True
>>> is_false('yes')
False
```

**Parameters**

- **v** (*Any*) – The value to check for falseyness
- **chk_none** (*bool*) – If `True`, treat `None`/`'none'`/`'null'` as Falsey (default `True`)

**Return bool is_False** `True` if the value appears to be falsey, otherwise `False`.

`privex.helpers.common.`**`is_true`**(*v*) → bool

Check if a given bool/str/int value is some form of `True`:

- **bool**: `True`
- **str**: `'true'`, `'yes'`, `'y'`, `'1'`
- **int**: `1`

(note: strings are automatically .lower()'d)

Usage:

```
>>> is_true('true')
True
>>> is_true('no')
False
```

**Parameters** **v** (*Any*) – The value to check for truthfulness

**Return bool is_true** `True` if the value appears to be truthy, otherwise `False`.

`privex.helpers.common.`**parse_csv**(*line: str*, *csvsplit: str = ','*) → List[str]

Quick n' dirty parsing of a simple comma separated line, with automatic whitespace stripping of both the `line` itself, and the values within the commas.

Example:

```
>>> parse_csv('  hello ,  world, test')
['hello', 'world', 'test']
>>> parse_csv(' world  ;   test   ; example', csvsplit=';')
['world', 'test', 'example']
```

**Parameters** **csvsplit** (*str*) – A character (or several) used to terminate each value in the list. Default: comma `,`

`privex.helpers.common.`**parse_keyval**(*line: str*, *valsplit: str = ':'*, *csvsplit=','*) → List[Tuple[str, str]]

Parses a csv with key:value pairs such as:

```
John Alex:Doe,Jane Sarah:Doe
```

Into a list with tuple pairs (can be easily converted to a dict):

```
[
    ('John Alex', 'Doe'),
    ('Jane Sarah', 'Doe')
]
```

By default, uses a colons `:` to split the key/value, and commas `,` to terminate the end of each keyval pair. This can be overridden by changing valsplit/csvsplit.

**Parameters**

- **line** (*str*) – A string of key:value pairs separated by commas e.g. `John Alex:Doe, Jane Sarah:Doe`
- **valsplit** (*str*) – A character (or several) used to split the key from the value (default: colon `:`)
- **csvsplit** (*str*) – A character (or several) used to terminate each keyval pair (default: comma `,`)

**Return List[Tuple[str,str]] parsed_data** A list of (key, value) tuples that can easily be casted to a dict()

privex.helpers.common.**random_str**(*size: int = 50*, *chars: Sequence = 'abcdefhkmnprstwxyz23456789ACDEFGHJKLMNPRSTWXYZ'*) → str

Generate a random string of arbitrary length using a given character set (string / list / tuple). Uses Python's SystemRandom class to provide relatively secure randomness from the OS. (On Linux, uses /dev/urandom)

By default, uses the character set `SAFE_CHARS` which contains letters a-z / A-Z and numbers 2-9 with commonly misread characters removed (such as `1`, `l`, `L`, `0` and `o`). Pass `ALPHANUM` as *chars* if you need the full set of upper/lowercase + numbers.

Usage:

```
>>> from privex.helpers import random_str
>>> # Default random string - 50 character alphanum without easily mistaken chars
>>> password = random_str()
'MrCWLYMYtT9A7bHc5ZNE4hn7PxHPmsWaT9GpfCkmZASK7ApN8r'
>>> # Customised random string - 12 characters using only the characters␣
↪`abcdef12345`
>>> custom = random_str(12, chars='abcdef12345')
'aba4cc14a43d'
```

Warning: As this relies on the OS's entropy features, it may not be cryptographically secure on non-Linux platforms:

> The returned data should be unpredictable enough for cryptographic applications, though its exact quality > depends on the OS implementation.

**Parameters**

- **size** (*int*) – Length of random string to generate (default 50 characters)
- **chars** (*str*) – Characterset to generate with ( default is `SAFE_CHARS` - a-z/A-Z/0-9 with often misread chars removed)

### 1.2.2 privex.helpers.decorators

Class Method / Function decorators

**Copyright**:

```
    +==================================================+
    |                 © 2019 Privex Inc.               |
    |               https://www.privex.io              |
    +==================================================+
    |                                                  |
    |        Originally Developed by Privex Inc.       |
    |                                                  |
    |        Core Developer(s):                        |
    |                                                  |
    |          (+)  Chris (@someguy123) [Privex]       |
    |          (+)  Kale (@kryogenic) [Privex]         |
    |                                                  |
    +==================================================+

Copyright 2019     Privex Inc.    ( https://www.privex.io )

Permission is hereby granted, free of charge, to any person obtaining a copy of
```

(continues on next page)

(continued from previous page)

```
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to use,
copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the
Software, and to permit persons to whom the Software is furnished to do so,
subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

## Functions

| | |
|---|---|
| *retry_on_err*([max_retries, delay]) | Decorates a function or class method, wraps the function/method with a try/catch block, and will automatically re-run the function with the same arguments up to *max_retries* time after any exception is raised, with a `delay` second delay between re-tries. |
| *r_cache*(cache_key[, cache_time, ...]) | This is a decorator which caches the result of the wrapped function into Redis using the key `cache_key` and with an expiry of `cache_time` seconds. |

### 1.2.2.1 privex.helpers.decorators.retry_on_err

privex.helpers.decorators.**retry_on_err**(*max_retries: int = 3*, *delay: int = 3*, ***retry_conf*)

Decorates a function or class method, wraps the function/method with a try/catch block, and will automatically re-run the function with the same arguments up to *max_retries* time after any exception is raised, with a `delay` second delay between re-tries.

If it still throws an exception after `max_retries` retries, it will log the exception details with `fail_msg`, and then re-raise it.

Usage (retry up to 5 times, 1 second between retries, stop immediately if IOError is detected):

```
>>> @retry_on_err(5, 1, fail_on=[IOError])
... def my_func(self, some=None, args=None):
...     if some == 'io': raise IOError()
...      raise FileExistsError()
```

This will be re-ran 5 times, 1 second apart after each exception is raised, before giving up:

```
>>> my_func()
```

Where-as this one will immediately re-raise the caught IOError on the first attempt, as it's passed in `fail_on`:

```
>>> my_func('io')
```

**Parameters**

- **max_retries** (*int*) – Maximum total retry attempts before giving up
- **delay** (*int*) – Amount of time in seconds to sleep before re-trying the wrapped function
- **retry_conf** – Less frequently used arguments, pass in as keyword args:

- (list) fail_on: A list() of Exception types that should result in immediate failure (don't retry, raise)
- (str) retry_msg: Override the log message used for retry attempts. First message param %s is func name, second message param %d is retry attempts remaining
- (str) fail_msg: Override the log message used after all retry attempts are exhausted. First message param %s is func name, and second param %d is amount of times retried.

#### 1.2.2.2 privex.helpers.decorators.r_cache

privex.helpers.decorators.**r_cache**(*cache_key: Union[str, callable], cache_time=300, format_args: list = None, format_opt: privex.helpers.decorators.FormatOpt = <FormatOpt.POS_AUTO: 'force_pos'>, **opts*) → Any

This is a decorator which caches the result of the wrapped function into Redis using the key `cache_key` and with an expiry of `cache_time` seconds.

Future calls to the wrapped function would then load the data from Redis until the cache expires, upon which it will re-run the original code and re-cache it.

To bypass the cache, pass kwarg `r_cache=False` to the wrapped function. To override the cache key on demand, pass `r_cache_key='mykey'` to the wrapped function.

**Example usage**:

```
>>> from privex.helpers import r_cache
>>>
>>> @r_cache('mydata', cache_time=600)
... def my_func(*args, **kwargs):
...     time.sleep(60)
...     return "done"
```

This will run the function and take 60 seconds to return while it sleeps

```
>>> my_func()
done
```

This will run instantly because "done" is now cached in Redis for 600 seconds

```
>>> my_func()
done
```

This will take another 60 seconds to run because `r_cache` is set to *False* (disables the cache)

```
>>> my_func(r_cache=False)
done
```

**Using a dynamic cache_key**:

**Simplest and most reliable - pass ''r_cache_key'' as an additional kwarg**

If you don't mind passing an additional kwarg to your function, then the most reliable method is to override the cache key by passing `r_cache_key` to your wrapped function.

Don't worry, we remove both `r_cache` and `r_cache_key` from the kwargs that actually hit your function.

```
>>> my_func(r_cache_key='somekey')     # Use the redis key 'somekey' when␣
↪caching data for this function
```

**Option 2. Pass a callable which takes the same arguments as the wrapped function**

In the example below, `who` takes two arguments: `name` and `title` - we then pass the function `make_key` which takes the same arguments - `r_cache` will detect that the cache key is a function and call it with the same `(*args, **kwargs)` passed to the wrapped function.

```
>>> from privex.helpers import r_cache
>>>
>>> def make_key(name, title):
...     return f"mycache:{name}"
...
>>> @r_cache(make_key)
... def who(name, title):
...     return "Their name is {title} {name}"
...
```

We can also obtain the same effect with a `lambda` callable defined directly inside of the cache_key.

```
>>> @r_cache(lambda name,title: f"mycache:{name}")
... def who(name, title):
...     return "Their name is {title} {name}"
```

**Option 3. Can be finnicky - using ‘‘format_args‘‘ to integrate with existing code**

If you can't change how your existing function/method is called, then you can use the `format_args` feature.

**NOTE:** Unless you're forcing the usage of kwargs with a function/method, it's strongly recommended that you keep `force_pos` enabled, and specify both the positional argument ID, and the kwarg name.

Basic Example:

```
>>> from privex.helpers import r_cache
>>> import time
>>>
>>> @r_cache('some_cache:{}:{}', cache_time=600, format_args=[0, 1, 'x',
↪'y'])
... def some_func(x=1, y=2):
...     time.sleep(5)
...     return 'x + y = {}'.format(x + y)
>>>
```

Using positional arguments, we can see from the debug log that it's formatting the `{}:{}` in the key with `x:y`

```
>>> some_func(1, 2)
2019-08-21 06:58:29,823 lg  DEBUG    Trying to load "some_cache:1:2" from␣
↪Redis cache
2019-08-21 06:58:29,826 lg  DEBUG    Not found in cache, or "r_cache" set␣
↪to false. Calling wrapped function.
'x + y = 3'
>>> some_func(2, 3)
```

(continues on next page)

(continued from previous page)

```
2019-08-21 06:58:34,831 lg  DEBUG    Trying to load "some_cache:2:3" from␣
↪Redis cache
2019-08-21 06:58:34,832 lg  DEBUG    Not found in cache, or "r_cache" set␣
↪to false. Calling wrapped function.
'x + y = 5'
```

When we passed `(1, 2)` and `(2, 3)` it had to re-run the function for each. But once we re-call it for the previously ran `(1, 2)` - it's able to retrieve the cached result just for those args.

```
>>> some_func(1, 2)
2019-08-21 06:58:41,752 lg  DEBUG    Trying to load "some_cache:1:2" from␣
↪Redis cache
'x + y = 3'
```

Be warned that the default format option `POS_AUTO` will make kwargs' values be specified in the same order as they were listed in `format_args`

```
>>> some_func(y=1, x=2)   # ``format_args`` has the kwargs in the order␣
↪``['x', 'y']`` thus ``.format(x,y)``
2019-08-21 06:58:58,611 lg  DEBUG    Trying to load "some_cache:2:1" from␣
↪Redis cache
2019-08-21 06:58:58,611 lg  DEBUG    Not found in cache, or "r_cache" set␣
↪to false. Calling wrapped function.
'x + y = 3'
```

**Parameters**

- **whitelist** (*bool*) – (default: `True`) If True, only use specified arg positions / kwarg keys when formatting `cache_key` placeholders. Otherwise, trust whatever args/kwargs were passed to the func.
- **format_opt** (`FormatOpt`) – (default: *FormatOpt.POS_AUTO*) "Format option" - how should args/kwargs be used when filling placeholders in the `cache_key` (see comments on FormatOption)
- **format_args** (*list*) – A list of positional arguments numbers (e.g. `[0, 1, 2]`) and/or kwargs `['x', 'y', 'z']` that should be used to format the *cache_key*
- **cache_key** (*str*) – The redis key to store the cached data into, e.g. *mydata*
- **cache_time** (*int*) – The amount of time in seconds to cache the result for (default: 300 seconds)

**Return Any res** The return result, either from the wrapped function, or from Redis.

### Classes

| | |
|---|---|
| *FormatOpt* | This enum represents various options available for *r_cache()* 's `format_opt` parameter. |

#### 1.2.2.3 privex.helpers.decorators.FormatOpt

**class** privex.helpers.decorators.**FormatOpt**

This enum represents various options available for *r_cache()* 's `format_opt` parameter.

To avoid bloating the PyDoc for `r_cache` too much, descriptions for each formatting option is available as a

short PyDoc comment under each enum option.

Usage:

```
>>> @r_cache('mykey', format_args=[0, 'x'], format_opt=FormatOpt.POS_AUTO)
```

**__init__**()
Initialize self. See help(type(self)) for accurate signature.

### Attributes

| | |
|---|---|
| *KWARG_ONLY* | Only use kwargs for formatting the cache key - requires named format placeholders, i.e. |
| *MIX* | Use both *args and **kwargs to format the cache_key (assuming mixed placeholders e.g.‘‘mykey:{}:{y}‘‘. |
| *POS_AUTO* | First attempt to format using *args whitelisted in `format_args`, if that causes a KeyError/IndexError, then pass kwarg values in the order they’re listed in `format_args` (only includes kwarg names listed in `format_args`) |
| *POS_ONLY* | Only use positional args for formatting the cache key, kwargs will be ignored completely. |

**KWARG_ONLY = 'kwarg'**
Only use kwargs for formatting the cache key - requires named format placeholders, i.e. `mykey:{x}`

**MIX = 'mix'**
Use both *args and **kwargs to format the cache_key (assuming mixed placeholders e.g. `mykey:{}:{y}`

**POS_AUTO = 'force_pos'**
First attempt to format using *args whitelisted in `format_args`, if that causes a KeyError/IndexError, then pass kwarg values in the order they’re listed in `format_args` (only includes kwarg names listed in `format_args`)

# def func(x, y) func(‘a’, ‘b’) # assuming 0 and 1 are in format_args, then it would use .format(‘a’, ‘b’) func(y=’b’, x=’a’) # assuming format_args = `['x','y']`, then it would use .format(‘a’, ‘b’)

**POS_ONLY = 'pos_only'**
Only use positional args for formatting the cache key, kwargs will be ignored completely.

privex.helpers.decorators.**FO**
alias of *privex.helpers.decorators.FormatOpt*

**class** privex.helpers.decorators.**FormatOpt**
This enum represents various options available for *r_cache()* ‘s `format_opt` parameter.

To avoid bloating the PyDoc for `r_cache` too much, descriptions for each formatting option is available as a short PyDoc comment under each enum option.

Usage:

```
>>> @r_cache('mykey', format_args=[0, 'x'], format_opt=FormatOpt.POS_AUTO)
```

**KWARG_ONLY = 'kwarg'**
Only use kwargs for formatting the cache key - requires named format placeholders, i.e. `mykey:{x}`

**MIX = 'mix'**
Use both *args and **kwargs to format the cache_key (assuming mixed placeholders e.g. `mykey:{}:{y}`

**POS_AUTO = 'force_pos'**
First attempt to format using *args whitelisted in `format_args`, if that causes a KeyError/IndexError, then pass kwarg values in the order they're listed in `format_args` (only includes kwarg names listed in `format_args`)

# def func(x, y) func('a', 'b') # assuming 0 and 1 are in format_args, then it would use .format('a', 'b') func(y='b', x='a') # assuming format_args = `['x','y']`, then it would use .format('a', 'b')

**POS_ONLY = 'pos_only'**
Only use positional args for formatting the cache key, kwargs will be ignored completely.

`privex.helpers.decorators.`**r_cache**(*cache_key: Union[str, callable], cache_time=300, format_args: list = None, format_opt: privex.helpers.decorators.FormatOpt = <FormatOpt.POS_AUTO: 'force_pos'>, **opts*) → Any

This is a decorator which caches the result of the wrapped function into Redis using the key `cache_key` and with an expiry of `cache_time` seconds.

Future calls to the wrapped function would then load the data from Redis until the cache expires, upon which it will re-run the original code and re-cache it.

To bypass the cache, pass kwarg `r_cache=False` to the wrapped function. To override the cache key on demand, pass `r_cache_key='mykey'` to the wrapped function.

**Example usage**:

```
>>> from privex.helpers import r_cache
>>>
>>> @r_cache('mydata', cache_time=600)
... def my_func(*args, **kwargs):
...     time.sleep(60)
...     return "done"
```

This will run the function and take 60 seconds to return while it sleeps

```
>>> my_func()
done
```

This will run instantly because "done" is now cached in Redis for 600 seconds

```
>>> my_func()
done
```

This will take another 60 seconds to run because `r_cache` is set to *False* (disables the cache)

```
>>> my_func(r_cache=False)
done
```

**Using a dynamic cache_key**:

**Simplest and most reliable - pass ''r_cache_key'' as an additional kwarg**

If you don't mind passing an additional kwarg to your function, then the most reliable method is to override the cache key by passing `r_cache_key` to your wrapped function.

Don't worry, we remove both `r_cache` and `r_cache_key` from the kwargs that actually hit your function.

```
>>> my_func(r_cache_key='somekey')     # Use the redis key 'somekey' when␣
↪caching data for this function
```

**Option 2. Pass a callable which takes the same arguments as the wrapped function**

In the example below, `who` takes two arguments: `name` and `title` - we then pass the function `make_key` which takes the same arguments - `r_cache` will detect that the cache key is a function and call it with the same `(*args, **kwargs)` passed to the wrapped function.

```
>>> from privex.helpers import r_cache
>>>
>>> def make_key(name, title):
...     return f"mycache:{name}"
...
>>> @r_cache(make_key)
... def who(name, title):
...     return "Their name is {title} {name}"
...
```

We can also obtain the same effect with a `lambda` callable defined directly inside of the cache_key.

```
>>> @r_cache(lambda name,title: f"mycache:{name}")
... def who(name, title):
...     return "Their name is {title} {name}"
```

**Option 3. Can be finnicky - using ‘‘format_args‘‘ to integrate with existing code**

If you can't change how your existing function/method is called, then you can use the `format_args` feature.

**NOTE:** Unless you're forcing the usage of kwargs with a function/method, it's strongly recommended that you keep `force_pos` enabled, and specify both the positional argument ID, and the kwarg name.

Basic Example:

```
>>> from privex.helpers import r_cache
>>> import time
>>>
>>> @r_cache('some_cache:{}:{}', cache_time=600, format_args=[0, 1, 'x',
↪'y'])
... def some_func(x=1, y=2):
...     time.sleep(5)
...     return 'x + y = {}'.format(x + y)
>>>
```

Using positional arguments, we can see from the debug log that it's formatting the `{}:{}` in the key with `x:y`

```
>>> some_func(1, 2)
2019-08-21 06:58:29,823 lg  DEBUG    Trying to load "some_cache:1:2" from␣
↪Redis cache
2019-08-21 06:58:29,826 lg  DEBUG    Not found in cache, or "r_cache" set␣
↪to false. Calling wrapped function.
'x + y = 3'
>>> some_func(2, 3)
2019-08-21 06:58:34,831 lg  DEBUG    Trying to load "some_cache:2:3" from␣
↪Redis cache
```

(continues on next page)

(continued from previous page)

```
2019-08-21 06:58:34,832 lg  DEBUG    Not found in cache, or "r_cache" set␣
↪to false. Calling wrapped function.
'x + y = 5'
```

When we passed `(1, 2)` and `(2, 3)` it had to re-run the function for each. But once we re-call it for the previously ran `(1, 2)` - it's able to retrieve the cached result just for those args.

```
>>> some_func(1, 2)
2019-08-21 06:58:41,752 lg  DEBUG    Trying to load "some_cache:1:2" from␣
↪Redis cache
'x + y = 3'
```

Be warned that the default format option `POS_AUTO` will make kwargs' values be specified in the same order as they were listed in `format_args`

```
>>> some_func(y=1, x=2)    # ``format_args`` has the kwargs in the order␣
↪``['x', 'y']`` thus ``.format(x,y)``
2019-08-21 06:58:58,611 lg  DEBUG    Trying to load "some_cache:2:1" from␣
↪Redis cache
2019-08-21 06:58:58,611 lg  DEBUG    Not found in cache, or "r_cache" set␣
↪to false. Calling wrapped function.
'x + y = 3'
```

**Parameters**

- **whitelist** (*bool*) – (default: `True`) If True, only use specified arg positions / kwarg keys when formatting `cache_key` placeholders. Otherwise, trust whatever args/kwargs were passed to the func.
- **format_opt** (`FormatOpt`) – (default: *FormatOpt.POS_AUTO*) "Format option" - how should args/kwargs be used when filling placeholders in the `cache_key` (see comments on FormatOption)
- **format_args** (*list*) – A list of positional arguments numbers (e.g. `[0, 1, 2]`) and/or kwargs `['x', 'y', 'z']` that should be used to format the *cache_key*
- **cache_key** (*str*) – The redis key to store the cached data into, e.g. *mydata*
- **cache_time** (*int*) – The amount of time in seconds to cache the result for (default: 300 seconds)

**Return Any res** The return result, either from the wrapped function, or from Redis.

`privex.helpers.decorators.`**retry_on_err**(*max_retries: int = 3*, *delay: int = 3*, ***retry_conf*)

Decorates a function or class method, wraps the function/method with a try/catch block, and will automatically re-run the function with the same arguments up to *max_retries* time after any exception is raised, with a `delay` second delay between re-tries.

If it still throws an exception after `max_retries` retries, it will log the exception details with `fail_msg`, and then re-raise it.

Usage (retry up to 5 times, 1 second between retries, stop immediately if IOError is detected):

```
>>> @retry_on_err(5, 1, fail_on=[IOError])
... def my_func(self, some=None, args=None):
...     if some == 'io': raise IOError()
...      raise FileExistsError()
```

This will be re-ran 5 times, 1 second apart after each exception is raised, before giving up:

```
>>> my_func()
```

Where-as this one will immediately re-raise the caught IOError on the first attempt, as it's passed in `fail_on`:

```
>>> my_func('io')
```

**Parameters**

- **max_retries** (*int*) – Maximum total retry attempts before giving up
- **delay** (*int*) – Amount of time in seconds to sleep before re-trying the wrapped function
- **retry_conf** – Less frequently used arguments, pass in as keyword args:

- (list) fail_on: A list() of Exception types that should result in immediate failure (don't retry, raise)
- (str) retry_msg: Override the log message used for retry attempts. First message param %s is func name, second message param %d is retry attempts remaining
- (str) fail_msg: Override the log message used after all retry attempts are exhausted. First message param %s is func name, and second param %d is amount of times retried.

### 1.2.3 privex.helpers.django

This module file contains Django-specific helper functions, to help save time when developing with the Django framework.

- handle_error - Redirects normal web page requests with a session error, outputs JSON with a status code for API queries.
- is_database_synchronized - Check if all migrations have been ran before running code.
- model_to_dict - Extract an individual Django model instance into a dict (with display names)
- to_json - Convert a model Queryset into a plain string JSON array with display names

**Copyright**:

```
    +==========================================================+
    |                 © 2019 Privex Inc.                       |
    |               https://www.privex.io                      |
    +==========================================================+
    |                                                          |
    |        Originally Developed by Privex Inc.               |
    |                                                          |
    |        Core Developer(s):                                |
    |                                                          |
    |          (+)  Chris (@someguy123) [Privex]               |
    |          (+)  Kale (@kryogenic) [Privex]                 |
    |                                                          |
    +==========================================================+

Copyright 2019     Privex Inc.   ( https://www.privex.io )

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to use,
```

(continues on next page)

(continued from previous page)

```
copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the
Software, and to permit persons to whom the Software is furnished to do so,
subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

### Functions

| | |
|---|---|
| *handle_error*(request, err, rdr[, status]) | Output an error as either a Django session message + redirect, or a JSON response based on whether the request was for the API readable version (?format=json) or not. |
| *is_database_synchronized*(database) | Check if all migrations have been ran. |
| *model_to_dict*(model) | 1 dimensional json-ifyer for any Model |
| *to_json*(query_set) | Iterate a Django query set and dump to json str |

#### 1.2.3.1 privex.helpers.django.handle_error

privex.helpers.django.**handle_error**(*request: django.http.request.HttpRequest*, *err: str*, *rdr: django.http.response.HttpResponseRedirectBase*, *status=400*)

Output an error as either a Django session message + redirect, or a JSON response based on whether the request was for the API readable version (?format=json) or not.

Usage:

```
>>> from django.shortcuts import redirect
>>> def my_view(request):
...     return handle_error(request, "Invalid password", redirect('/login'), 403)
```

**Parameters**

- **request** (*HttpRequest*) – The Django request object from your view
- **err** (*str*) – An error message as a string to display to the user / api call
- **rdr** (*HttpResponseRedirectBase*) – A redirect() for normal browsers to follow after adding the session error.
- **status** (*int*) – The HTTP status code to return if the request is an API call (default: 400 bad request)

#### 1.2.3.2 privex.helpers.django.is_database_synchronized

privex.helpers.django.**is_database_synchronized**(*database: str*) → bool

Check if all migrations have been ran. Useful for preventing auto-running code accessing models before the tables even exist, thus preventing you from migrating...

```
>>> from django.db import DEFAULT_DB_ALIAS
>>> if not is_database_synchronized(DEFAULT_DB_ALIAS):
>>>     log.warning('Cannot run reload_handlers because there are unapplied␣
↪migrations!')
>>>     return
```

**Parameters** **database** (*str*) – Which Django database config is being used? Generally just pass django.db.DEFAULT_DB_ALIAS

**Return bool** True if all migrations have been ran, False if not.

#### 1.2.3.3 privex.helpers.django.model_to_dict

privex.helpers.django.**model_to_dict**(*model*) → dict

1 dimensional json-ifyer for any Model

#### 1.2.3.4 privex.helpers.django.to_json

privex.helpers.django.**to_json**(*query_set*) → str

Iterate a Django query set and dump to json str

privex.helpers.django.**handle_error**(*request: django.http.request.HttpRequest*, *err: str*, *rdr: django.http.response.HttpResponseRedirectBase*, *status=400*)

Output an error as either a Django session message + redirect, or a JSON response based on whether the request was for the API readable version (?format=json) or not.

Usage:

```
>>> from django.shortcuts import redirect
>>> def my_view(request):
...     return handle_error(request, "Invalid password", redirect('/login'), 403)
```

**Parameters**

- **request** (*HttpRequest*) – The Django request object from your view
- **err** (*str*) – An error message as a string to display to the user / api call
- **rdr** (*HttpResponseRedirectBase*) – A redirect() for normal browsers to follow after adding the session error.
- **status** (*int*) – The HTTP status code to return if the request is an API call (default: 400 bad request)

privex.helpers.django.**is_database_synchronized**(*database: str*) → bool

Check if all migrations have been ran. Useful for preventing auto-running code accessing models before the tables even exist, thus preventing you from migrating...

```
>>> from django.db import DEFAULT_DB_ALIAS
>>> if not is_database_synchronized(DEFAULT_DB_ALIAS):
>>>     log.warning('Cannot run reload_handlers because there are unapplied␣
↪migrations!')
>>>     return
```

**Parameters** **database** (*str*) – Which Django database config is being used? Generally just pass django.db.DEFAULT_DB_ALIAS

**Return bool** True if all migrations have been ran, False if not.

privex.helpers.django.**model_to_dict**(*model*) → dict
1 dimensional json-ifyer for any Model

privex.helpers.django.**to_json**(*query_set*) → str
Iterate a Django query set and dump to json str

### 1.2.4 privex.helpers.exceptions

Exception classes used either by our helpers, or just generic exception names which are missing from the standard base exceptions in Python, and are commonly used across our projects.

**Copyright**:

```
    +==================================================+
    |                 © 2019 Privex Inc.               |
    |               https://www.privex.io              |
    +==================================================+
    |                                                  |
    |        Originally Developed by Privex Inc.       |
    |                                                  |
    |        Core Developer(s):                        |
    |                                                  |
    |          (+)  Chris (@someguy123) [Privex]       |
    |          (+)  Kale (@kryogenic) [Privex]         |
    |                                                  |
    +==================================================+

Copyright 2019     Privex Inc.   ( https://www.privex.io )

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to use,
copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the
Software, and to permit persons to whom the Software is furnished to do so,
subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

### Exceptions

| | |
|---|---|
| *BaseDNSException* | Base exception for DNS-related exceptions |
| *BoundaryException* | Thrown when the v4/v6 address boundary for a reverse DNS record is invalid. |
| *DomainNotFound* | Thrown when a (sub)domain or it's parent(s) could not be found |
| *InvalidDNSRecord* | Thrown when a passed DNS record is not valid |
| *PrivexException* | Base exception for all custom Privex exceptions |

#### 1.2.4.1 privex.helpers.exceptions.BaseDNSException

**exception** privex.helpers.exceptions.**BaseDNSException**
Base exception for DNS-related exceptions

#### 1.2.4.2 privex.helpers.exceptions.BoundaryException

**exception** privex.helpers.exceptions.**BoundaryException**
Thrown when the v4/v6 address boundary for a reverse DNS record is invalid.

An "address boundary" is the minimum amount of bits that defines a character in a reverse DNS domain (i.e. those ending in in-addr/ip6.arpa):

- IPv6 uses "nibbles" of 4 bits for each individual character in a standard 128-bit address
- IPv4 uses "octets" of 8 bits, which represent 0-255 in each of the 4 "blocks" within an IPv4 address

This means the minimum and maximum boundaries for IPv4 and IPv6 are as such:

IPv6 - Minimum boundary: 4 bits (`*.f.ip6.arpa`), Maximum boundary: 128 bits (rDNS for a single address, i.e. a /128) IPv4 - Minimum boundary: 8 bits (`x.x.x.127.in-addr.arpa`), Maximum boundary: 32 bits (`1.0.0.127.in-addr.arpa`)

#### 1.2.4.3 privex.helpers.exceptions.DomainNotFound

**exception** privex.helpers.exceptions.**DomainNotFound**
Thrown when a (sub)domain or it's parent(s) could not be found

#### 1.2.4.4 privex.helpers.exceptions.InvalidDNSRecord

**exception** privex.helpers.exceptions.**InvalidDNSRecord**
Thrown when a passed DNS record is not valid

#### 1.2.4.5 privex.helpers.exceptions.PrivexException

**exception** privex.helpers.exceptions.**PrivexException**
Base exception for all custom Privex exceptions

**exception** privex.helpers.exceptions.**BaseDNSException**
Base exception for DNS-related exceptions

**exception** privex.helpers.exceptions.**BoundaryException**
Thrown when the v4/v6 address boundary for a reverse DNS record is invalid.

An "address boundary" is the minimum amount of bits that defines a character in a reverse DNS domain (i.e. those ending in in-addr/ip6.arpa):

- IPv6 uses "nibbles" of 4 bits for each individual character in a standard 128-bit address
- IPv4 uses "octets" of 8 bits, which represent 0-255 in each of the 4 "blocks" within an IPv4 address

This means the minimum and maximum boundaries for IPv4 and IPv6 are as such:

IPv6 - Minimum boundary: 4 bits (`*.f.ip6.arpa`), Maximum boundary: 128 bits (rDNS for a single address, i.e. a /128) IPv4 - Minimum boundary: 8 bits (`x.x.x.127.in-addr.arpa`), Maximum boundary: 32 bits (`1.0.0.127.in-addr.arpa`)

**exception** privex.helpers.exceptions.**DomainNotFound**
Thrown when a (sub)domain or it's parent(s) could not be found

**exception** privex.helpers.exceptions.**InvalidDNSRecord**
Thrown when a passed DNS record is not valid

**exception** privex.helpers.exceptions.**PrivexException**
Base exception for all custom Privex exceptions

### 1.2.5 privex.helpers.net

Network related helper code

**Copyright**:

```
    +===================================================+
    |                 © 2019 Privex Inc.                |
    |               https://www.privex.io               |
    +===================================================+
    |                                                   |
    |        Originally Developed by Privex Inc.        |
    |                                                   |
    |        Core Developer(s):                         |
    |                                                   |
    |          (+)  Chris (@someguy123) [Privex]        |
    |          (+)  Kale (@kryogenic) [Privex]          |
    |                                                   |
    +===================================================+

Copyright 2019     Privex Inc.   ( https://www.privex.io )

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to use,
copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the
Software, and to permit persons to whom the Software is furnished to do so,
subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
```

(continues on next page)

(continued from previous page)

```
PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

### Functions

| | |
|---|---|
| *asn_to_name*(as_number[, quiet]) | Look up an integer Autonomous System Number and return the human readable name of the organization. |
| *ip4_to_rdns*(ip_obj[, v4_boundary, boundary]) | Internal function for getting the rDNS domain for a given v4 address. |
| *ip6_to_rdns*(ip_obj[, v6_boundary, boundary]) | Internal function for getting the rDNS domain for a given v6 address. |
| *ip_is_v4*(ip) | Determines whether an IP address is IPv4 or not |
| *ip_is_v6*(ip) | Determines whether an IP address is IPv6 or not |
| *ip_to_rdns*(ip[, boundary, v6_boundary, ...]) | Converts an IPv4 or IPv6 address into an in-addr domain |

#### 1.2.5.1 privex.helpers.net.asn_to_name

privex.helpers.net.**asn_to_name**(*as_number: Union[int, str], quiet: bool = True*) → str

Look up an integer Autonomous System Number and return the human readable name of the organization.

Usage:

```
>>> asn_to_name(210083)
'PRIVEX, SE'
>>> asn_to_name('13335')
'CLOUDFLARENET - Cloudflare, Inc., US'
```

This helper function requires `dnspython>=1.16.0`, it will not be visible unless you install the dnspython package in your virtualenv, or systemwide:

```
pip3 install dnspython
```

**Parameters**

- **as_number** (*int/str*) – The AS number as a string or integer, e.g. 210083 or '210083'
- **quiet** (*bool*) – (default True) If True, returns 'Unknown ASN' if a lookup fails. If False, raises a KeyError if no results are found.

**Raises** **KeyError** – Raised when a lookup returns no results, and `quiet` is set to False.

**Return str as_name** The name and country code of the ASN, e.g. 'PRIVEX, SE'

#### 1.2.5.2 privex.helpers.net.ip4_to_rdns

privex.helpers.net.**ip4_to_rdns**(*ip_obj: ipaddress.IPv4Address, v4_boundary: int = 24, boundary: bool = False*) → str

Internal function for getting the rDNS domain for a given v4 address. Use *ip_to_rdns()* unless you have a specific need for this one.

**Parameters**

- **ip_obj** (*IPv4Address*) – An IPv4 ip_address() object to get the rDNS domain for
- **v4_boundary** (*int*) – 8-32 bits. If `boundary` is True, return the base rDNS domain at this boundary.
- **boundary** (*bool*) – If True, cut off the rDNS domain to the given `v4_boundary`

**Return str rdns_domain** in-addr.arpa format, e.g. `0.0.127.in-addr.arpa`

#### 1.2.5.3 privex.helpers.net.ip6_to_rdns

`privex.helpers.net.`**ip6_to_rdns**(*ip_obj: ipaddress.IPv6Address*, *v6_boundary: int = 32*, *boundary: bool = False*) → str

Internal function for getting the rDNS domain for a given v6 address. Use *ip_to_rdns()* unless you have a specific need for this one.

**Parameters**

- **ip_obj** (*IPv6Address*) – An IPv4 ip_address() object to get the rDNS domain for
- **v4_boundary** (*int*) – 8-128 bits. If `boundary` is True, return the base rDNS domain at this boundary.
- **boundary** (*bool*) – If True, cut off the rDNS domain to the given `v6_boundary`

**Return str rdns_domain** ip6.arpa format, e.g. `0.8.e.f.ip6.arpa`

#### 1.2.5.4 privex.helpers.net.ip_is_v4

`privex.helpers.net.`**ip_is_v4**(*ip: str*) → bool

Determines whether an IP address is IPv4 or not

**Parameters** **ip** (*str*) – An IP address as a string, e.g. 192.168.1.1

**Raises** **ValueError** – When the given IP address `ip` is invalid

**Return bool** True if IPv6, False if not (i.e. probably IPv4)

#### 1.2.5.5 privex.helpers.net.ip_is_v6

`privex.helpers.net.`**ip_is_v6**(*ip: str*) → bool

Determines whether an IP address is IPv6 or not

**Parameters** **ip** (*str*) – An IP address as a string, e.g. 192.168.1.1

**Raises** **ValueError** – When the given IP address `ip` is invalid

**Return bool** True if IPv6, False if not (i.e. probably IPv4)

#### 1.2.5.6 privex.helpers.net.ip_to_rdns

`privex.helpers.net.`**ip_to_rdns**(*ip: str*, *boundary: bool = False*, *v6_boundary: int = 32*, *v4_boundary: int = 24*) → str

Converts an IPv4 or IPv6 address into an in-addr domain

Default boundaries: IPv4 - 24 bits, IPv6 - 32 bits

**Examples:**

```
>>> ip_to_rdns('127.0.0.1') # IPv4 to arpa format
    '1.0.0.127.in-addr.arpa'
```

```
>>> ip_to_rdns('2001:dead:beef::1') # IPv6 to arpa format
    '1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.f.e.e.b.d.a.e.d.1.0.0.2.ip6.arpa'
```

```
>>> ip_to_rdns('2001:dead:beef::1', boundary=True) # IPv6 32-bit boundary to arpa
    'd.a.e.d.1.0.0.2.ip6.arpa'
```

**Parameters**

- **ip** (*str*) – IPv4 or IPv6 address
- **boundary** (*bool*) – If True, return the base (boundary) domain to place NS/SOA
- **v6_boundary** (*int*) – Bits for IPv6 boundary. Must be dividable by 4 bits (nibble)
- **v4_boundary** (*int*) – Bits for IPv4 boundary. Must be dividable by 8 bits (octet)

**Raises**

- **ValueError** – When IP address is invalid
- ***BoundaryException*** – When boundary for IPv4/v6 is invalid

**Return str rdns_domain** in-addr.arpa format, e.g. `0.0.127.in-addr.arpa`

**Return str rdns_domain** ip6.arpa format, e.g. `0.8.e.f.ip6.arpa`

`privex.helpers.net.`**asn_to_name**(*as_number: Union[int, str]*, *quiet: bool = True*) → str

Look up an integer Autonomous System Number and return the human readable name of the organization.

Usage:

```
>>> asn_to_name(210083)
'PRIVEX, SE'
>>> asn_to_name('13335')
'CLOUDFLARENET - Cloudflare, Inc., US'
```

This helper function requires `dnspython>=1.16.0`, it will not be visible unless you install the dnspython package in your virtualenv, or systemwide:

```
pip3 install dnspython
```

**Parameters**

- **as_number** (*int/str*) – The AS number as a string or integer, e.g. 210083 or '210083'
- **quiet** (*bool*) – (default True) If True, returns 'Unknown ASN' if a lookup fails. If False, raises a KeyError if no results are found.

**Raises** **KeyError** – Raised when a lookup returns no results, and `quiet` is set to False.

**Return str as_name** The name and country code of the ASN, e.g. 'PRIVEX, SE'

`privex.helpers.net.`**ip4_to_rdns**(*ip_obj: ipaddress.IPv4Address*, *v4_boundary: int = 24*, *boundary: bool = False*) → str

Internal function for getting the rDNS domain for a given v4 address. Use *ip_to_rdns()* unless you have a specific need for this one.

**Parameters**

- **ip_obj** (*IPv4Address*) – An IPv4 ip_address() object to get the rDNS domain for
- **v4_boundary** (*int*) – 8-32 bits. If `boundary` is True, return the base rDNS domain at this boundary.
- **boundary** (*bool*) – If True, cut off the rDNS domain to the given `v4_boundary`

**Return str rdns_domain** in-addr.arpa format, e.g. `0.0.127.in-addr.arpa`

`privex.helpers.net.`**ip6_to_rdns**(*ip_obj: ipaddress.IPv6Address*, *v6_boundary: int = 32*, *boundary: bool = False*) → str

Internal function for getting the rDNS domain for a given v6 address. Use *ip_to_rdns()* unless you have a specific need for this one.

**Parameters**

- **ip_obj** (*IPv6Address*) – An IPv4 ip_address() object to get the rDNS domain for
- **v4_boundary** (*int*) – 8-128 bits. If `boundary` is True, return the base rDNS domain at this boundary.
- **boundary** (*bool*) – If True, cut off the rDNS domain to the given `v6_boundary`

**Return str rdns_domain** ip6.arpa format, e.g. `0.8.e.f.ip6.arpa`

`privex.helpers.net.`**ip_is_v4**(*ip: str*) → bool

Determines whether an IP address is IPv4 or not

**Parameters** **ip** (*str*) – An IP address as a string, e.g. 192.168.1.1

**Raises** **ValueError** – When the given IP address `ip` is invalid

**Return bool** True if IPv6, False if not (i.e. probably IPv4)

`privex.helpers.net.`**ip_is_v6**(*ip: str*) → bool

Determines whether an IP address is IPv6 or not

**Parameters** **ip** (*str*) – An IP address as a string, e.g. 192.168.1.1

**Raises** **ValueError** – When the given IP address `ip` is invalid

**Return bool** True if IPv6, False if not (i.e. probably IPv4)

`privex.helpers.net.`**ip_to_rdns**(*ip: str*, *boundary: bool = False*, *v6_boundary: int = 32*, *v4_boundary: int = 24*) → str

Converts an IPv4 or IPv6 address into an in-addr domain

Default boundaries: IPv4 - 24 bits, IPv6 - 32 bits

**Examples:**

```
>>> ip_to_rdns('127.0.0.1') # IPv4 to arpa format
    '1.0.0.127.in-addr.arpa'
```

```
>>> ip_to_rdns('2001:dead:beef::1') # IPv6 to arpa format
    '1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.f.e.e.b.d.a.e.d.1.0.0.2.ip6.arpa'
```

```
>>> ip_to_rdns('2001:dead:beef::1', boundary=True) # IPv6 32-bit boundary to arpa
    'd.a.e.d.1.0.0.2.ip6.arpa'
```

**Parameters**

- **ip** (*str*) – IPv4 or IPv6 address
- **boundary** (*bool*) – If True, return the base (boundary) domain to place NS/SOA

- **v6_boundary** (*int*) – Bits for IPv6 boundary. Must be dividable by 4 bits (nibble)
- **v4_boundary** (*int*) – Bits for IPv4 boundary. Must be dividable by 8 bits (octet)

**Raises**

- **ValueError** – When IP address is invalid
- ***BoundaryException*** – When boundary for IPv4/v6 is invalid

**Return str rdns_domain** in-addr.arpa format, e.g. `0.0.127.in-addr.arpa`

**Return str rdns_domain** ip6.arpa format, e.g. `0.8.e.f.ip6.arpa`

### 1.2.6 privex.helpers.plugin

This module handles connection objects for databases, APIs etc. by exposing functions which initialise and store class instances for re-use.

It's primarily intended to be used to enable database, caching and third-party API connectivity for the helpers in this package, however, you're free to use the functions / classes / attributes exposed in this module for your own apps.

Classes are generally initialised using the settings from *settings* - see the docs for that module to learn how to override the settings if the defaults don't work for you.

**Copyright**:

### Functions

| | |
|---|---|
| *get_redis*() | Get a Redis connection object. |

privex.helpers.plugin.**HAS_REDIS = True**
If the redis module was imported successfully, this will change to True.

privex.helpers.plugin.**get_redis**() → redis.client.Redis
Get a Redis connection object. Create one if it doesn't exist.

## 1.2.7 privex.helpers.settings

Configuration options for helpers, and services they depend on, such as Redis.

To override settings from your app:

```
>>> from privex.helpers import settings
>>> settings.REDIS_HOST = 'redis.example.org'
>>> settings.REDIS_PORT = 1234
```

**Copyright**:

```
    +==================================================+
    |                 © 2019 Privex Inc.               |
    |               https://www.privex.io              |
    +==================================================+
    |                                                  |
    |        Originally Developed by Privex Inc.       |
    |                                                  |
    |        Core Developer(s):                        |
    |                                                  |
    |          (+)  Chris (@someguy123) [Privex]       |
    |          (+)  Kale (@kryogenic) [Privex]         |
    |                                                  |
    +==================================================+

Copyright 2019     Privex Inc.   ( https://www.privex.io )

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to use,
copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the
Software, and to permit persons to whom the Software is furnished to do so,
subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

privex.helpers.settings.**REDIS_DB = 0**
Redis database to use (number)

privex.helpers.settings.**REDIS_HOST = 'localhost'**
Hostname / IP address where redis-server is running on

privex.helpers.settings.**REDIS_PORT = 6379**
Port number that Redis is running on at REDIS_HOST

## 1.3 Unit Tests

| | |
|---|---|
| *tests* | This file contains test cases for Privex's Python Helper's (privex-helpers). |

### 1.3.1 tests

This file contains test cases for Privex's Python Helper's (privex-helpers).

Before running the tests:

- Ensure you have any mandatory requirements installed (see setup.py's install_requires)
- You may wish to install any optional requirements listed in README.md for best results
- Python 3.7 is recommended at the time of writing this. See README.md in-case this has changed.

To run the tests, simply execute `./tests.py` in your shell:

```
user@the-matrix ~/privex-helpers $ ./tests.py
.........................
----------------------------------------------------------------------
Ran 28 tests in 0.001s

OK
```

If for some reason you don't have the executable `python3.7` in your PATH, try running by hand with `python3`

```
user@the-matrix ~/privex-helpers $ python3 tests.py
.........................
----------------------------------------------------------------------
Ran 28 tests in 0.001s

OK
```

For more verbosity, simply add `-v` to the end of the command:

```
user@the-matrix ~/privex-helpers $ ./tests.py -v
test_empty_combined (__main__.TestBoolHelpers) ... ok
test_isfalse_truthy (__main__.TestBoolHelpers) ... ok
test_v4_arpa_boundary_16bit (__main__.TestIPReverseDNS)
Test generating 16-bit v4 boundary ... ok
test_v4_arpa_boundary_24bit (__main__.TestIPReverseDNS)
Test generating 24-bit v4 boundary ... ok
test_kval_single (__main__.TestParseHelpers)
Test that a single value still returns a list ... ok
test_kval_spaced (__main__.TestParseHelpers)
```

(continues on next page)

(continued from previous page)

```
Test key:val csv parsing with excess outer whitespace, and value whitespace ... ok
# Truncated excess output in this PyDoc example, as there are many more lines showing
# the results of each individual testcase, wasting space and adding bloat...
----------------------------------------------------------------------
Ran 28 tests in 0.001s

OK
```

You can also use the `pytest` tool (used by default for our Travis CI):

```
user@host: ~/privex-helpers $ pip3 install pytest
# You can add `-v` for more detailed output, just like when running tests.py directly.
user@host: ~/privex-helpers $ pytest tests.py

====================================== test session starts␣
↪======================================
platform darwin -- Python 3.7.0, pytest-5.0.1, py-1.8.0, pluggy-0.12.0
rootdir: /home/user/privex-helpers
collected 33 items

tests.py ..............................                                  ␣
↪  [100%]

=================================== warnings summary␣
↪====================================
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/jinja2/
↪utils.py:485
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/jinja2/
↪utils.py:485:
DeprecationWarning: Using or importing the ABCs from 'collections' instead of from
↪'collections.abc'
is deprecated, and in 3.8 it will stop working
    from collections import MutableMapping
============================ 33 passed, 2 warnings in 0.17 seconds␣
↪============================
```

**Copyright**:

```
Copyright 2019         Privex Inc.    ( https://www.privex.io )
License: X11 / MIT     Github: https://github.com/Privex/python-helpers
```

### Classes

| | |
|---|---|
| *EmptyIter* | A mock iterable object with zero length for testing empty() |
| *TestBoolHelpers*([methodName]) | Test the boolean check functions is_true, is_false, as well as empty() |
| *TestIPReverseDNS*([methodName]) | Unit testing for the reverse DNS functions in *privex.helpers.net* |
| *TestParseHelpers*([methodName]) | Test the parsing functions parse_csv and parse_keyval |
| *PrivexBaseCase*([methodName]) | Base test-case for module test cases to inherit. |

#### 1.3.1.1 tests.EmptyIter

**class** tests.**EmptyIter**
A mock iterable object with zero length for testing empty()

**__init__**()
Initialize self. See help(type(self)) for accurate signature.

##### Methods

| | |
|---|---|
| *__init__* | Initialize self. |

#### 1.3.1.2 tests.TestBoolHelpers

**class** tests.**TestBoolHelpers**(*methodName='runTest'*)
Test the boolean check functions is_true, is_false, as well as empty()

**__init__**(*methodName='runTest'*)
Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

##### Methods

| | |
|---|---|
| *__init__*([methodName]) | Create an instance of the class that will use the named test method when executed. |
| addCleanup(function, *args, **kwargs) | Add a function, with arguments, to be called when the test is completed. |
| addTypeEqualityFunc(typeobj, function) | Add a type specific assertEqual style function to compare a type. |
| assertAlmostEqual(first, second[, places, ...]) | Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the difference between the two objects is more than the given delta. |
| assertAlmostEquals(**kwargs) | |
| assertCountEqual(first, second[, msg]) | An unordered sequence comparison asserting that the same elements, regardless of order. |
| assertDictContainsSubset(subset, dictionary) | Checks whether dictionary is a superset of subset. |
| assertDictEqual(d1, d2[, msg]) | |
| assertEqual(first, second[, msg]) | Fail if the two objects are unequal as determined by the '==' operator. |
| assertEquals(**kwargs) | |
| assertFalse(expr[, msg]) | Check that the expression is false. |
| assertGreater(a, b[, msg]) | Just like self.assertTrue(a > b), but with a nicer default message. |
| assertGreaterEqual(a, b[, msg]) | Just like self.assertTrue(a >= b), but with a nicer default message. |
| assertIn(member, container[, msg]) | Just like self.assertTrue(a in b), but with a nicer default message. |

Continued on next page

Table 14 – continued from previous page

| | |
|---|---|
| assertIs(expr1, expr2[, msg]) | Just like self.assertTrue(a is b), but with a nicer default message. |
| assertIsInstance(obj, cls[, msg]) | Same as self.assertTrue(isinstance(obj, cls)), with a nicer default message. |
| assertIsNone(obj[, msg]) | Same as self.assertTrue(obj is None), with a nicer default message. |
| assertIsNot(expr1, expr2[, msg]) | Just like self.assertTrue(a is not b), but with a nicer default message. |
| assertIsNotNone(obj[, msg]) | Included for symmetry with assertIsNone. |
| assertLess(a, b[, msg]) | Just like self.assertTrue(a < b), but with a nicer default message. |
| assertLessEqual(a, b[, msg]) | Just like self.assertTrue(a <= b), but with a nicer default message. |
| assertListEqual(list1, list2[, msg]) | A list-specific equality assertion. |
| assertLogs([logger, level]) | Fail unless a log message of level *level* or higher is emitted on *logger_name* or its children. |
| assertMultiLineEqual(first, second[, msg]) | Assert that two multi-line strings are equal. |
| assertNotAlmostEqual(first, second[, ...]) | Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the difference between the two objects is less than the given delta. |
| assertNotAlmostEquals(**kwargs) | |
| assertNotEqual(first, second[, msg]) | Fail if the two objects are equal as determined by the '!=' operator. |
| assertNotEquals(**kwargs) | |
| assertNotIn(member, container[, msg]) | Just like self.assertTrue(a not in b), but with a nicer default message. |
| assertNotIsInstance(obj, cls[, msg]) | Included for symmetry with assertIsInstance. |
| assertNotRegex(text, unexpected_regex[, msg]) | Fail the test if the text matches the regular expression. |
| assertNotRegexpMatches(**kwargs) | |
| assertRaises(expected_exception, *args, **kwargs) | Fail unless an exception of class expected_exception is raised by the callable when invoked with specified positional and keyword arguments. |
| assertRaisesRegex(expected_exception, ...) | Asserts that the message in a raised exception matches a regex. |
| assertRaisesRegexp(**kwargs) | |
| assertRegex(text, expected_regex[, msg]) | Fail the test unless the text matches the regular expression. |
| assertRegexpMatches(**kwargs) | |
| assertSequenceEqual(seq1, seq2[, msg, seq_type]) | An equality assertion for ordered sequences (like lists and tuples). |
| assertSetEqual(set1, set2[, msg]) | A set-specific equality assertion. |
| assertTrue(expr[, msg]) | Check that the expression is true. |
| assertTupleEqual(tuple1, tuple2[, msg]) | A tuple-specific equality assertion. |
| assertWarns(expected_warning, *args, **kwargs) | Fail unless a warning of class warnClass is triggered by the callable when invoked with specified positional and keyword arguments. |
| assertWarnsRegex(expected_warning, ...) | Asserts that the message in a triggered warning matches a regexp. |

Continued on next page

Table 14 – continued from previous page

| | |
|---|---|
| assert_(**kwargs) | |
| countTestCases() | |
| debug() | Run the test without collecting errors in a TestResult |
| defaultTestResult() | |
| doCleanups() | Execute all cleanup functions. |
| fail([msg]) | Fail immediately, with the given message. |
| failIf(**kwargs) | |
| failIfAlmostEqual(**kwargs) | |
| failIfEqual(**kwargs) | |
| failUnless(**kwargs) | |
| failUnlessAlmostEqual(**kwargs) | |
| failUnlessEqual(**kwargs) | |
| failUnlessRaises(**kwargs) | |
| id() | |
| run([result]) | |
| setUp() | Hook method for setting up the test fixture before exercising it. |
| setUpClass() | Hook method for setting up class fixture before running tests in the class. |
| shortDescription() | Returns a one-line description of the test, or None if no description has been provided. |
| skipTest(reason) | Skip this test. |
| subTest([msg]) | Return a context manager that will return the enclosed block of code in a subtest identified by the optional message and keyword parameters. |
| tearDown() | Hook method for deconstructing the test fixture after testing it. |
| tearDownClass() | Hook method for deconstructing the class fixture after running all tests in the class. |
| test_empty_combined() | |
| test_empty_lst() | |
| test_empty_vals() | |
| test_empty_zero() | |
| test_isfalse_falsey() | |
| test_isfalse_truthy() | |
| test_istrue_falsey() | |
| test_istrue_truthy() | |
| test_notempty() | |

### Attributes

| |
|---|
| empty_lst |
| empty_vals |
| empty_zero |
| falsey |
| longMessage |
| maxDiff |
| truthy |

#### 1.3.1.3 tests.TestIPReverseDNS

**class** tests.**TestIPReverseDNS**(*methodName='runTest'*)

Unit testing for the reverse DNS functions in *privex.helpers.net*

**Covers:**

- positive resolution tests (generate standard rDNS domain from clean input)
- positive boundary tests (confirm valid results with range of boundaries)
- negative address tests (ensure errors thrown for invalid v4/v6 addresses)
- negative boundary tests (ensure errors thrown for invalid v4/v6 rDNS boundaries)

**__init__**(*methodName='runTest'*)

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

##### Methods

| | |
|---|---|
| *__init__*([methodName]) | Create an instance of the class that will use the named test method when executed. |
| addCleanup(function, *args, **kwargs) | Add a function, with arguments, to be called when the test is completed. |
| addTypeEqualityFunc(typeobj, function) | Add a type specific assertEqual style function to compare a type. |
| assertAlmostEqual(first, second[, places, ...]) | Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the difference between the two objects is more than the given delta. |
| assertAlmostEquals(**kwargs) | |
| assertCountEqual(first, second[, msg]) | An unordered sequence comparison asserting that the same elements, regardless of order. |
| assertDictContainsSubset(subset, dictionary) | Checks whether dictionary is a superset of subset. |
| assertDictEqual(d1, d2[, msg]) | |
| assertEqual(first, second[, msg]) | Fail if the two objects are unequal as determined by the '==' operator. |
| assertEquals(**kwargs) | |
| assertFalse(expr[, msg]) | Check that the expression is false. |
| assertGreater(a, b[, msg]) | Just like self.assertTrue(a > b), but with a nicer default message. |
| assertGreaterEqual(a, b[, msg]) | Just like self.assertTrue(a >= b), but with a nicer default message. |
| assertIn(member, container[, msg]) | Just like self.assertTrue(a in b), but with a nicer default message. |
| assertIs(expr1, expr2[, msg]) | Just like self.assertTrue(a is b), but with a nicer default message. |
| assertIsInstance(obj, cls[, msg]) | Same as self.assertTrue(isinstance(obj, cls)), with a nicer default message. |
| assertIsNone(obj[, msg]) | Same as self.assertTrue(obj is None), with a nicer default message. |

Continued on next page

Table 16 – continued from previous page

| | |
|---|---|
| assertIsNot(expr1, expr2[, msg]) | Just like self.assertTrue(a is not b), but with a nicer default message. |
| assertIsNotNone(obj[, msg]) | Included for symmetry with assertIsNone. |
| assertLess(a, b[, msg]) | Just like self.assertTrue(a < b), but with a nicer default message. |
| assertLessEqual(a, b[, msg]) | Just like self.assertTrue(a <= b), but with a nicer default message. |
| assertListEqual(list1, list2[, msg]) | A list-specific equality assertion. |
| assertLogs([logger, level]) | Fail unless a log message of level *level* or higher is emitted on *logger_name* or its children. |
| assertMultiLineEqual(first, second[, msg]) | Assert that two multi-line strings are equal. |
| assertNotAlmostEqual(first, second[, ...]) | Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the difference between the two objects is less than the given delta. |
| assertNotAlmostEquals(**kwargs) | |
| assertNotEqual(first, second[, msg]) | Fail if the two objects are equal as determined by the '!=' operator. |
| assertNotEquals(**kwargs) | |
| assertNotIn(member, container[, msg]) | Just like self.assertTrue(a not in b), but with a nicer default message. |
| assertNotIsInstance(obj, cls[, msg]) | Included for symmetry with assertIsInstance. |
| assertNotRegex(text, unexpected_regex[, msg]) | Fail the test if the text matches the regular expression. |
| assertNotRegexpMatches(**kwargs) | |
| assertRaises(expected_exception, *args, **kwargs) | Fail unless an exception of class expected_exception is raised by the callable when invoked with specified positional and keyword arguments. |
| assertRaisesRegex(expected_exception, ...) | Asserts that the message in a raised exception matches a regex. |
| assertRaisesRegexp(**kwargs) | |
| assertRegex(text, expected_regex[, msg]) | Fail the test unless the text matches the regular expression. |
| assertRegexpMatches(**kwargs) | |
| assertSequenceEqual(seq1, seq2[, msg, seq_type]) | An equality assertion for ordered sequences (like lists and tuples). |
| assertSetEqual(set1, set2[, msg]) | A set-specific equality assertion. |
| assertTrue(expr[, msg]) | Check that the expression is true. |
| assertTupleEqual(tuple1, tuple2[, msg]) | A tuple-specific equality assertion. |
| assertWarns(expected_warning, *args, **kwargs) | Fail unless a warning of class warnClass is triggered by the callable when invoked with specified positional and keyword arguments. |
| assertWarnsRegex(expected_warning, ...) | Asserts that the message in a triggered warning matches a regexp. |
| assert_(**kwargs) | |
| countTestCases() | |
| debug() | Run the test without collecting errors in a TestResult |
| defaultTestResult() | |
| doCleanups() | Execute all cleanup functions. |
| fail([msg]) | Fail immediately, with the given message. |

Continued on next page

Table 16 – continued from previous page

| | |
|---|---|
| failIf(**kwargs) | |
| failIfAlmostEqual(**kwargs) | |
| failIfEqual(**kwargs) | |
| failUnless(**kwargs) | |
| failUnlessAlmostEqual(**kwargs) | |
| failUnlessEqual(**kwargs) | |
| failUnlessRaises(**kwargs) | |
| id() | |
| run([result]) | |
| setUp() | Hook method for setting up the test fixture before exercising it. |
| setUpClass() | Hook method for setting up class fixture before running tests in the class. |
| shortDescription() | Returns a one-line description of the test, or None if no description has been provided. |
| skipTest(reason) | Skip this test. |
| subTest([msg]) | Return a context manager that will return the enclosed block of code in a subtest identified by the optional message and keyword parameters. |
| tearDown() | Hook method for deconstructing the test fixture after testing it. |
| tearDownClass() | Hook method for deconstructing the class fixture after running all tests in the class. |
| *test_v4_arpa_boundary_16bit*() | Test generating 16-bit v4 boundary |
| *test_v4_arpa_boundary_24bit*() | Test generating 24-bit v4 boundary |
| *test_v4_inv_boundary*() | Raise if IPv4 boundary isn't divisable by 8 |
| *test_v4_inv_boundary_2*() | Raise if IPv4 boundary is too short |
| *test_v4_invalid*() | Raise if IPv4 address has < 4 octets |
| *test_v4_invalid_2*() | Raise if IPv4 address has octet out of range |
| *test_v4_to_arpa*() | Test generating rDNS for standard v4 |
| *test_v6_arpa_boundary_16bit*() | Test generating 16-bit v6 boundary |
| *test_v6_arpa_boundary_32bit*() | Test generating 32-bit v6 boundary |
| *test_v6_inv_boundary*() | Raise if IPv6 boundary isn't dividable by 4 |
| *test_v6_inv_boundary_2*() | Raise if IPv6 boundary is too short |
| *test_v6_invalid*() | Raise if IPv6 address has invalid block formatting |
| *test_v6_invalid_2*() | Raise if v6 address has invalid chars |
| *test_v6_to_arpa*() | Test generating rDNS for standard v6 |

### Attributes

| |
|---|
| longMessage |
| maxDiff |

**test_v4_arpa_boundary_16bit**()
Test generating 16-bit v4 boundary

**test_v4_arpa_boundary_24bit**()
Test generating 24-bit v4 boundary

**test_v4_inv_boundary**()
Raise if IPv4 boundary isn't divisable by 8

**test_v4_inv_boundary_2**()
Raise if IPv4 boundary is too short

**test_v4_invalid**()
Raise if IPv4 address has < 4 octets

**test_v4_invalid_2**()
Raise if IPv4 address has octet out of range

**test_v4_to_arpa**()
Test generating rDNS for standard v4

**test_v6_arpa_boundary_16bit**()
Test generating 16-bit v6 boundary

**test_v6_arpa_boundary_32bit**()
Test generating 32-bit v6 boundary

**test_v6_inv_boundary**()
Raise if IPv6 boundary isn't dividable by 4

**test_v6_inv_boundary_2**()
Raise if IPv6 boundary is too short

**test_v6_invalid**()
Raise if IPv6 address has invalid block formatting

**test_v6_invalid_2**()
Raise if v6 address has invalid chars

**test_v6_to_arpa**()
Test generating rDNS for standard v6

#### 1.3.1.4 tests.TestParseHelpers

**class** tests.**TestParseHelpers**(*methodName='runTest'*)
Test the parsing functions parse_csv and parse_keyval

**__init__**(*methodName='runTest'*)
Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

##### Methods

| | |
|---|---|
| *__init__*([methodName]) | Create an instance of the class that will use the named test method when executed. |
| addCleanup(function, *args, **kwargs) | Add a function, with arguments, to be called when the test is completed. |
| addTypeEqualityFunc(typeobj, function) | Add a type specific assertEqual style function to compare a type. |
| assertAlmostEqual(first, second[, places, ...]) | Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the difference between the two objects is more than the given delta. |
| assertAlmostEquals(**kwargs) | |

Continued on next page

Table 18 – continued from previous page

| | |
|---|---|
| assertCountEqual(first, second[, msg]) | An unordered sequence comparison asserting that the same elements, regardless of order. |
| assertDictContainsSubset(subset, dictionary) | Checks whether dictionary is a superset of subset. |
| assertDictEqual(d1, d2[, msg]) | |
| assertEqual(first, second[, msg]) | Fail if the two objects are unequal as determined by the '==' operator. |
| assertEquals(**kwargs) | |
| assertFalse(expr[, msg]) | Check that the expression is false. |
| assertGreater(a, b[, msg]) | Just like self.assertTrue(a > b), but with a nicer default message. |
| assertGreaterEqual(a, b[, msg]) | Just like self.assertTrue(a >= b), but with a nicer default message. |
| assertIn(member, container[, msg]) | Just like self.assertTrue(a in b), but with a nicer default message. |
| assertIs(expr1, expr2[, msg]) | Just like self.assertTrue(a is b), but with a nicer default message. |
| assertIsInstance(obj, cls[, msg]) | Same as self.assertTrue(isinstance(obj, cls)), with a nicer default message. |
| assertIsNone(obj[, msg]) | Same as self.assertTrue(obj is None), with a nicer default message. |
| assertIsNot(expr1, expr2[, msg]) | Just like self.assertTrue(a is not b), but with a nicer default message. |
| assertIsNotNone(obj[, msg]) | Included for symmetry with assertIsNone. |
| assertLess(a, b[, msg]) | Just like self.assertTrue(a < b), but with a nicer default message. |
| assertLessEqual(a, b[, msg]) | Just like self.assertTrue(a <= b), but with a nicer default message. |
| assertListEqual(list1, list2[, msg]) | A list-specific equality assertion. |
| assertLogs([logger, level]) | Fail unless a log message of level *level* or higher is emitted on *logger_name* or its children. |
| assertMultiLineEqual(first, second[, msg]) | Assert that two multi-line strings are equal. |
| assertNotAlmostEqual(first, second[, ...]) | Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the difference between the two objects is less than the given delta. |
| assertNotAlmostEquals(**kwargs) | |
| assertNotEqual(first, second[, msg]) | Fail if the two objects are equal as determined by the '!=' operator. |
| assertNotEquals(**kwargs) | |
| assertNotIn(member, container[, msg]) | Just like self.assertTrue(a not in b), but with a nicer default message. |
| assertNotIsInstance(obj, cls[, msg]) | Included for symmetry with assertIsInstance. |
| assertNotRegex(text, unexpected_regex[, msg]) | Fail the test if the text matches the regular expression. |
| assertNotRegexpMatches(**kwargs) | |
| assertRaises(expected_exception, *args, **kwargs) | Fail unless an exception of class expected_exception is raised by the callable when invoked with specified positional and keyword arguments. |

Continued on next page

Table 18 – continued from previous page

| | |
|---|---|
| assertRaisesRegex(expected_exception, ...) | Asserts that the message in a raised exception matches a regex. |
| assertRaisesRegexp(**kwargs) | |
| assertRegex(text, expected_regex[, msg]) | Fail the test unless the text matches the regular expression. |
| assertRegexpMatches(**kwargs) | |
| assertSequenceEqual(seq1, seq2[, msg, seq_type]) | An equality assertion for ordered sequences (like lists and tuples). |
| assertSetEqual(set1, set2[, msg]) | A set-specific equality assertion. |
| assertTrue(expr[, msg]) | Check that the expression is true. |
| assertTupleEqual(tuple1, tuple2[, msg]) | A tuple-specific equality assertion. |
| assertWarns(expected_warning, *args, **kwargs) | Fail unless a warning of class warnClass is triggered by the callable when invoked with specified positional and keyword arguments. |
| assertWarnsRegex(expected_warning, ...) | Asserts that the message in a triggered warning matches a regexp. |
| assert_(**kwargs) | |
| countTestCases() | |
| debug() | Run the test without collecting errors in a TestResult |
| defaultTestResult() | |
| doCleanups() | Execute all cleanup functions. |
| fail([msg]) | Fail immediately, with the given message. |
| failIf(**kwargs) | |
| failIfAlmostEqual(**kwargs) | |
| failIfEqual(**kwargs) | |
| failUnless(**kwargs) | |
| failUnlessAlmostEqual(**kwargs) | |
| failUnlessEqual(**kwargs) | |
| failUnlessRaises(**kwargs) | |
| id() | |
| run([result]) | |
| setUp() | Hook method for setting up the test fixture before exercising it. |
| setUpClass() | Hook method for setting up class fixture before running tests in the class. |
| shortDescription() | Returns a one-line description of the test, or None if no description has been provided. |
| skipTest(reason) | Skip this test. |
| subTest([msg]) | Return a context manager that will return the enclosed block of code in a subtest identified by the optional message and keyword parameters. |
| tearDown() | Hook method for deconstructing the test fixture after testing it. |
| tearDownClass() | Hook method for deconstructing the class fixture after running all tests in the class. |
| *test_csv_single*() | Test that a single value still returns a list |
| *test_csv_spaced*() | Test csv parsing with excess outer whitespace, and value whitespace |
| *test_kval_clean*() | Test that a clean key:val csv is parsed correctly |
| *test_kval_single*() | Test that a single value still returns a list |

Continued on next page

Table 18 – continued from previous page

| | |
|---|---|
| *test_kval_spaced*() | Test key:val csv parsing with excess outer whitespace, and value whitespace |

#### Attributes

| |
|---|
| longMessage |
| maxDiff |

**test_csv_single**()
Test that a single value still returns a list

**test_csv_spaced**()
Test csv parsing with excess outer whitespace, and value whitespace

**test_env_bool_false**()
Test env_bool returns False boolean with valid env var

**test_env_bool_true**()
Test env_bool returns True boolean with valid env var

**test_env_nonexist_bool**()
Test env_bool returns default with non-existant env var

**test_kval_clean**()
Test that a clean key:val csv is parsed correctly

**test_kval_custom_clean**()
Test that a clean key:val csv with custom split characters is parsed correctly (pipe for kv, semi-colon for pair separation)

**test_kval_custom_spaced**()
Test key:val csv parsing with excess outer/value whitespace, and custom split characters.

**test_kval_single**()
Test that a single value still returns a list

**test_kval_spaced**()
Test key:val csv parsing with excess outer whitespace, and value whitespace

#### 1.3.1.5 tests.PrivexBaseCase

**class** tests.**PrivexBaseCase**(*methodName='runTest'*)
Base test-case for module test cases to inherit.

Contains useful class attributes such as `falsey` and `empty_vals` that are used across different unit tests.

**__init__**(*methodName='runTest'*)
Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

#### Methods

| | |
|---|---|
| `__init__`([methodName]) | Create an instance of the class that will use the named test method when executed. |
| `addCleanup`(function, *args, **kwargs) | Add a function, with arguments, to be called when the test is completed. |
| `addTypeEqualityFunc`(typeobj, function) | Add a type specific assertEqual style function to compare a type. |
| `assertAlmostEqual`(first, second[, places, ...]) | Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the difference between the two objects is more than the given delta. |
| `assertAlmostEquals`(**kwargs) | |
| `assertCountEqual`(first, second[, msg]) | An unordered sequence comparison asserting that the same elements, regardless of order. |
| `assertDictContainsSubset`(subset, dictionary) | Checks whether dictionary is a superset of subset. |
| `assertDictEqual`(d1, d2[, msg]) | |
| `assertEqual`(first, second[, msg]) | Fail if the two objects are unequal as determined by the '==' operator. |
| `assertEquals`(**kwargs) | |
| `assertFalse`(expr[, msg]) | Check that the expression is false. |
| `assertGreater`(a, b[, msg]) | Just like self.assertTrue(a > b), but with a nicer default message. |
| `assertGreaterEqual`(a, b[, msg]) | Just like self.assertTrue(a >= b), but with a nicer default message. |
| `assertIn`(member, container[, msg]) | Just like self.assertTrue(a in b), but with a nicer default message. |
| `assertIs`(expr1, expr2[, msg]) | Just like self.assertTrue(a is b), but with a nicer default message. |
| `assertIsInstance`(obj, cls[, msg]) | Same as self.assertTrue(isinstance(obj, cls)), with a nicer default message. |
| `assertIsNone`(obj[, msg]) | Same as self.assertTrue(obj is None), with a nicer default message. |
| `assertIsNot`(expr1, expr2[, msg]) | Just like self.assertTrue(a is not b), but with a nicer default message. |
| `assertIsNotNone`(obj[, msg]) | Included for symmetry with assertIsNone. |
| `assertLess`(a, b[, msg]) | Just like self.assertTrue(a < b), but with a nicer default message. |
| `assertLessEqual`(a, b[, msg]) | Just like self.assertTrue(a <= b), but with a nicer default message. |
| `assertListEqual`(list1, list2[, msg]) | A list-specific equality assertion. |
| `assertLogs`([logger, level]) | Fail unless a log message of level *level* or higher is emitted on *logger_name* or its children. |
| `assertMultiLineEqual`(first, second[, msg]) | Assert that two multi-line strings are equal. |
| `assertNotAlmostEqual`(first, second[, ...]) | Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the difference between the two objects is less than the given delta. |
| `assertNotAlmostEquals`(**kwargs) | |

Continued on next page

Table 20 – continued from previous page

| | |
|---|---|
| assertNotEqual(first, second[, msg]) | Fail if the two objects are equal as determined by the '!=' operator. |
| assertNotEquals(**kwargs) | |
| assertNotIn(member, container[, msg]) | Just like self.assertTrue(a not in b), but with a nicer default message. |
| assertNotIsInstance(obj, cls[, msg]) | Included for symmetry with assertIsInstance. |
| assertNotRegex(text, unexpected_regex[, msg]) | Fail the test if the text matches the regular expression. |
| assertNotRegexpMatches(**kwargs) | |
| assertRaises(expected_exception, *args, **kwargs) | Fail unless an exception of class expected_exception is raised by the callable when invoked with specified positional and keyword arguments. |
| assertRaisesRegex(expected_exception, ...) | Asserts that the message in a raised exception matches a regex. |
| assertRaisesRegexp(**kwargs) | |
| assertRegex(text, expected_regex[, msg]) | Fail the test unless the text matches the regular expression. |
| assertRegexpMatches(**kwargs) | |
| assertSequenceEqual(seq1, seq2[, msg, seq_type]) | An equality assertion for ordered sequences (like lists and tuples). |
| assertSetEqual(set1, set2[, msg]) | A set-specific equality assertion. |
| assertTrue(expr[, msg]) | Check that the expression is true. |
| assertTupleEqual(tuple1, tuple2[, msg]) | A tuple-specific equality assertion. |
| assertWarns(expected_warning, *args, **kwargs) | Fail unless a warning of class warnClass is triggered by the callable when invoked with specified positional and keyword arguments. |
| assertWarnsRegex(expected_warning, ...) | Asserts that the message in a triggered warning matches a regexp. |
| assert_(**kwargs) | |
| countTestCases() | |
| debug() | Run the test without collecting errors in a TestResult |
| defaultTestResult() | |
| doCleanups() | Execute all cleanup functions. |
| fail([msg]) | Fail immediately, with the given message. |
| failIf(**kwargs) | |
| failIfAlmostEqual(**kwargs) | |
| failIfEqual(**kwargs) | |
| failUnless(**kwargs) | |
| failUnlessAlmostEqual(**kwargs) | |
| failUnlessEqual(**kwargs) | |
| failUnlessRaises(**kwargs) | |
| id() | |
| run([result]) | |
| setUp() | Hook method for setting up the test fixture before exercising it. |
| setUpClass() | Hook method for setting up class fixture before running tests in the class. |
| shortDescription() | Returns a one-line description of the test, or None if no description has been provided. |
| skipTest(reason) | Skip this test. |

Continued on next page

Table 20 – continued from previous page

| | |
|---|---|
| subTest([msg]) | Return a context manager that will return the enclosed block of code in a subtest identified by the optional message and keyword parameters. |
| tearDown() | Hook method for deconstructing the test fixture after testing it. |
| tearDownClass() | Hook method for deconstructing the class fixture after running all tests in the class. |

### Attributes

| | |
|---|---|
| empty_lst | |
| empty_vals | |
| empty_zero | |
| *falsey* | Normal False-y values, as various types |
| *falsey_empty* | False-y values, plus 'empty' values like '' and None |
| longMessage | |
| maxDiff | |
| *truthy* | Truthful values, as various types |

**falsey = ['false', 'FALSE', False, 0, '0', 'no']**
Normal False-y values, as various types

**falsey_empty = ['false', 'FALSE', False, 0, '0', 'no', None, '', 'null']**
False-y values, plus 'empty' values like '' and None

**truthy = [True, 'TRUE', 'true', 'yes', 'y', '1', 1]**
Truthful values, as various types

**class** tests.**EmptyIter**
A mock iterable object with zero length for testing empty()

**class** tests.**PrivexBaseCase**(*methodName='runTest'*)
Base test-case for module test cases to inherit.

Contains useful class attributes such as falsey and empty_vals that are used across different unit tests.

**falsey = ['false', 'FALSE', False, 0, '0', 'no']**
Normal False-y values, as various types

**falsey_empty = ['false', 'FALSE', False, 0, '0', 'no', None, '', 'null']**
False-y values, plus 'empty' values like '' and None

**truthy = [True, 'TRUE', 'true', 'yes', 'y', '1', 1]**
Truthful values, as various types

**class** tests.**TestBoolHelpers**(*methodName='runTest'*)
Test the boolean check functions is_true, is_false, as well as empty()

**class** tests.**TestIPReverseDNS**(*methodName='runTest'*)
Unit testing for the reverse DNS functions in *privex.helpers.net*

**Covers:**

- positive resolution tests (generate standard rDNS domain from clean input)
- positive boundary tests (confirm valid results with range of boundaries)
- negative address tests (ensure errors thrown for invalid v4/v6 addresses)

- negative boundary tests (ensure errors thrown for invalid v4/v6 rDNS boundaries)

**test_v4_arpa_boundary_16bit**()
Test generating 16-bit v4 boundary

**test_v4_arpa_boundary_24bit**()
Test generating 24-bit v4 boundary

**test_v4_inv_boundary**()
Raise if IPv4 boundary isn't divisable by 8

**test_v4_inv_boundary_2**()
Raise if IPv4 boundary is too short

**test_v4_invalid**()
Raise if IPv4 address has < 4 octets

**test_v4_invalid_2**()
Raise if IPv4 address has octet out of range

**test_v4_to_arpa**()
Test generating rDNS for standard v4

**test_v6_arpa_boundary_16bit**()
Test generating 16-bit v6 boundary

**test_v6_arpa_boundary_32bit**()
Test generating 32-bit v6 boundary

**test_v6_inv_boundary**()
Raise if IPv6 boundary isn't dividable by 4

**test_v6_inv_boundary_2**()
Raise if IPv6 boundary is too short

**test_v6_invalid**()
Raise if IPv6 address has invalid block formatting

**test_v6_invalid_2**()
Raise if v6 address has invalid chars

**test_v6_to_arpa**()
Test generating rDNS for standard v6

**class** tests.**TestParseHelpers**(*methodName='runTest'*)
Test the parsing functions parse_csv and parse_keyval

**test_csv_single**()
Test that a single value still returns a list

**test_csv_spaced**()
Test csv parsing with excess outer whitespace, and value whitespace

**test_env_bool_false**()
Test env_bool returns False boolean with valid env var

**test_env_bool_true**()
Test env_bool returns True boolean with valid env var

**test_env_nonexist_bool**()
Test env_bool returns default with non-existant env var

**test_kval_clean**()
Test that a clean key:val csv is parsed correctly

**test_kval_custom_clean**()
Test that a clean key:val csv with custom split characters is parsed correctly (pipe for kv, semi-colon for pair separation)

**test_kval_custom_spaced**()
Test key:val csv parsing with excess outer/value whitespace, and custom split characters.

**test_kval_single**()
Test that a single value still returns a list

**test_kval_spaced**()
Test key:val csv parsing with excess outer whitespace, and value whitespace

**class** tests.**TestRedisCache**(*methodName='runTest'*)
Unit tests which verify that the decorator `privex.helpers.decorators.r_test()` caches correctly, and also verifies dynamic cache key generation works as expected.

**setUp**()
Hook method for setting up the test fixture before exercising it.

**tearDown**()
Remove any Redis keys used during test, to avoid failure on re-run

**test_rcache_callable**()
Decorate random string function - use a lambda callable to determine a cache key

**test_rcache_rand**()
Decorate random string function with r_cache - test that two calls return the same string

**test_rcache_rand_dynamic**()
Decorate random string function with r_cache and use format_args for dynamic cache string testing

CHAPTER

TWO

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## p



## t

# INDEX

## Symbols



## A



## B



## D



## E



## F



## G



## H



## I



## K



## M



## P