Privex Python Helpers Documentation

Privex Inc., Chris (Someguy123)

Jun 12, 2020

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

Contents

CHAPTER

ONE

QUICK INSTALL

Installing with Pipenv **(recommended)**

```
pipenv install privex-helpers
```

Installing with standard `pip3`

```
pip3 install privex-helpers
```

CHAPTER

TWO

PYTHON MODULE OVERVIEW

Privex's Python Helpers is organised into various sub-modules to make it easier to find the functions/classes you want to use, and to avoid having to load the entire module (though it's lightweight).

With the exception of *privex.helpers.django* (Django gets upset if certain django modules are imported before it's initialised), **all functions/classes are imported within the `__init__` file,** allowing you to simply type:

```
from privex.helpers import empty, run_sync, asn_to_name
```

Instead of having to import the functions from each individual module:

```
from privex.helpers.common import empty
from privex.helpers.asyncx import run_sync
from privex.helpers.net import asn_to_name
```

Below is a listing of the sub-modules available in `privex-helpers` with a short description of what each module contains.

privex.helpers.asyncx	Functions and classes related to working with Python's native asyncio support
privex.helpers.black_magic	This module contains *somewhat risky* code that uses app introspection e.g.
privex.helpers.cache	Helper functions/classes related to caching.
privex.helpers.common	Common functions and classes that don't fit into a specific category
privex.helpers.collections	Functions, classes and/or types which either **are**, or are related to Python variable storage types (`dict`, `tuple`, `list`, `set` etc.)
privex.helpers.converters	Various functions/classes which convert/parse objects from one type into another.
privex.helpers.crypto	Cryptography related helper classes/functions
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.
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.extras	Various helper functions/classes which depend on a certain package being installed.
privex.helpers.net	Network related helper code

continues on next page

Table 1 – continued from previous page

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.
privex.helpers.setuppy	Helpers for setup.py, e.g.
privex.helpers.types	

CHAPTER

THREE

ALL DOCUMENTATION

3.1 Installation

3.1.1 Download and install from PyPi using pip (recommended)

```
pip3 install privex-helpers
```

3.1.2 (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
```

3.2 Example Usages

3.2.1 Boolean testing

3.2.1.1 The `empty` function

The `empty()` function in our opinion, is one of most useful functions in this library. It allows for a clean, readable method of checking if a variable is "empty", e.g. when checking keyword arguments to a function.

With a single argument, it simply tests if a variable is `""` (empty string) or `None`.

The argument `itr` can be set to `True` if you consider an empty iterable such as `[]` or `{}` as "empty". This functionality also supports objects which implement `__len__`, and also checks to ensure `__len__` is available, avoiding an exception if an object doesn't support it.

The argument `zero` can be set to `True` if you want to consider `0` (integer) and `'0'` (string) as "empty".

```
from privex.helpers import empty

x, y = "", None
z, a = [], 0

empty(x) # True
empty(y) # True
empty(z) # False
empty(z, itr=True) # True
empty(a) # False
empty(a, zero=True) # True
```

3.2.1.2 The `is_true` and `is_false` functions

When handling user input, whether from an environment file (`.env`), or from data passed to a web API, it can be a pain attempting to check for booleans.

A boolean `True` could be represented as the string `'true'`, `'1'`, `'YES'`, as an integer `1`, or even an actual boolean `True`. Trying to test for all of those cases requires a rather long `if` statement...

Thus *`is_true()`* and *`is_false()`* were created.

```
from privex.helpers import is_true, is_false

is_true(0)          # False
is_true(1)          # True
is_true('1')        # True
is_true('true')     # True
is_true('false')    # False
is_true('orange')   # False
is_true('YeS')      # True

is_false(0)         # True
is_false('false')   # True
is_false('true')    # False
is_false(False)     # True
```

3.2.2 Handling environmental variables in different formats

3.2.2.1 Using `env_csv` to support lists contained within an env var

The function *`env_csv()`* parses a CSV-like environment variable into a list

```
from privex.helpers import env_csv
import os
os.environ['EXAMPLE'] = "this,   is, an,example   "

env_csv('EXAMPLE', ['error'])
# returns: ['this', 'is', 'an', 'example']
```

(continues on next page)

(continued from previous page)

```
env_csv('NOEXIST', ['non-existent'])
# returns: ['non-existent']
```

3.2.2.2 Using `env_keyval` to support dictionaries contained within an env var

The function *`env_keyval()`* parses an environment variable into a ordered list of tuple pairs, which can be easily converted into a dictionary using `dict()`.

```
from privex.helpers import env_keyval
import os
os.environ['EXAMPLE'] = "John:  Doe , Jane   : Doe, Aaron:Smith"

env_keyval('EXAMPLE')
# returns: [('John', 'Doe'), ('Jane', 'Doe'), ('Aaron', 'Smith')]
env_keyval('NOEXIST', {})
# returns: {}
```

3.2.3 Improved collections, including dict's and namedtuple's

In our *`privex.helpers.collections`* module (plus maybe a few things in *`privex.helpers.common`*), we have various functions and classes designed to make working with Python's storage types more painless, while trying to keep compatibility with code that expects the native types.

3.2.3.1 Dictionaries with dot notation attribute read/write

Dictionaries (`dict`) are powerful, and easy to deal with. But why can't you read or write dictionary items with attribute dot notation!?

This is where *`DictObject`* comes in to save the day. It's a child class of python's native `dict` type, which means it's still compatible with functions/methods such as `json.dumps()`, and in most cases will be plug-n-play with existing dict-using code.

Basic usage

```
from privex.helpers import DictObject

x = dict(hello='world', lorem='ipsum')
x['hello']  # This works with a normal dict
x.hello     # But this raises: AttributeError: 'dict' object has no attribute 'hello'

# We can cast the dict 'x' into a DictObject
y = DictObject(x)
y['hello']          # Returns: 'world'
y.hello             # Returns: 'world'

# Not only can you access dict keys via attributes, you can also set keys via␣
↪attributes
y.example = 'testing'
y                   # We can see below that setting 'example' worked as expected.
# Output: {'hello': 'world', 'lorem': 'ipsum', 'example': 'testing'}
```

Type checking / Equality comparison

As *DictObject* is a subclass of `dict`, you can use `isinstance()` to check against `dict` (e.g. `isinstance(DictObject(), dict)`) and it should return True.

You can also compare dictionary equality between a *DictObject* and a `dict` using == as normal.

```
y = DictObject(hello='world')

isinstance(y, dict)    # You should always use isinstance instead of `type(x) == dict`
# Returns: True

# You can also use typing.Dict with isinstance when checking a DictObject
from typing import Dict
isinstance(y, Dict)    # Returns: True

# You can compare equality between a DictObject and a dict with no problems
DictObject(hello='world') == dict(hello='world')
# Returns: True
DictObject(hello='world') == dict(hello='example')
# Returns: False
```

Type Masquerading

Also included is the class `MockDictObj`, which is a subclass of *DictObject* with it's name, qualified name, and module adjusted so that it appears to be the builtin `dict` type.

This may help in some cases, but sadly can't fool a `type(x) == dict` check.

```
from privex.helpers import MockDictObj
z = MockDictObj(y)
type(z)                    # Returns: <class 'dict'>
z.__class__.__module__     # Returns: 'builtins'
```

3.2.3.2 Named Tuple's (namedtuple) with dict-like key access, dict casting, and writable fields

A somewhat simpler version of `dict`'s are `collections.namedtuple()`'s

Unfortunately they have a few quirks that can make them annoying to deal with.

```
Person = namedtuple('Person', 'first_name last_name')  # This is an existing␣
↪namedtuple "type" or "class"
john = Person('John', 'Doe')  # This is an existing namedtuple instance
john.first_name               # This works on a standard namedtuple. Returns: John
john[1]                       # This works on a standard namedtuple. Returns: Doe
john['first_name']            # However, this would throw a TypeError.
dict(john)                    # This would throw a ValueError.
john.address = '123 Fake St'  # This raises an AttributeError.
```

Thus, we created *dictable_namedtuple()* (and more), which creates namedtuples with additional functionality, including item/key access of fields, easy casting into dictionaries, and ability to add new fields.

```
from privex.helpers import dictable_namedtuple
Person = dictable_namedtuple('Person', 'first_name last_name')
john = Person('John', 'Doe')
dave = Person(first_name='Dave', last_name='Smith')
print(dave['first_name'])       # Prints:  Dave
print(dave.first_name)          # Prints:  Dave
print(john[1])                  # Prints:  Doe
print(dict(john))               # Prints:  {'first_name': 'John', 'last_name': 'Doe'}
```

(continues on next page)

(continued from previous page)

```
john.address = '123 Fake St'     # Unlike normal namedtuple, we can add new fields
print(john)                      # Prints: Person(first_name='John', last_name='Doe',␣
↪address='123 Fake St')
```

You can use *convert_dictable_namedtuple()* to convert existing `namedtuple` instancess into `dictable_namedtuple` instances:

```
Person = namedtuple('Person', 'first_name last_name')  # This is an existing␣
↪namedtuple "type" or "class"
john = Person('John', 'Doe')  # This is an existing namedtuple instance

d_john = convert_dictable_namedtuple(john)
d_john.first_name              # Returns: John
d_john[1]                      # Returns: Doe
d_john['first_name']           # Returns: 'John'
dict(d_john)                   # Returns: {'first_name': 'John', 'last_name': 'Doe'}
```

For more information, check out the module docs at *privex.helpers.collections*

privex.helpers.asyncx	Functions and classes related to working with Python's native asyncio support
privex.helpers.black_magic	This module contains *somewhat risky* code that uses app introspection e.g.
privex.helpers.cache	Helper functions/classes related to caching.
privex.helpers.common	Common functions and classes that don't fit into a specific category
privex.helpers.collections	Functions, classes and/or types which either **are**, or are related to Python variable storage types (`dict`, `tuple`, `list`, `set` etc.)
privex.helpers.converters	Various functions/classes which convert/parse objects from one type into another.
privex.helpers.crypto	Cryptography related helper classes/functions
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.
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.extras	Various helper functions/classes which depend on a certain package being installed.
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.
privex.helpers.setuppy	Helpers for setup.py, e.g.
privex.helpers.types	

3.3 privex.helpers.asyncx

Functions and classes related to working with Python's native asyncio support

To avoid issues with the `async` keyword, this file is named `asyncx` instead of `async`

Copyright:

Attributes

—

Functions

async_sync(f)	Async Synchronous Decorator, borrowed from https://stackoverflow.com/a/23036785/2648583 - added this PyDoc comment and support for returning data from a synchronous function
await_if_needed(func, *args, **kwargs)	Call, await, and/or simply return `func` depending on whether it's an async function reference (coroutine function), a non-awaited coroutine, a standard synchronous function, or just a plain old string.
awaitable(func)	Decorator which helps with creation of async wrapper functions.
awaitable_class(cls)	Wraps a class, allowing all async methods to be used in non-async code as if they were normal synchronous methods.
call_sys_async(proc, *args[, write])	Async version of *call_sys()* - works exactly the same, other than needing to be `await`'d.
get_async_type(obj)	Detects if `obj` is an async object/function that needs awaited / called, whether it's a synchronous callable, or whether it's unknown (probably not async)
is_async_context()	Returns `True` if currently in an async context, otherwise `False`

continues on next page

Table 3 – continued from previous page

loop_run(coro, *args[, _loop])	Run the coroutine or async function `coro` synchronously, using an AsyncIO event loop.
run_sync(func, *args, **kwargs)	Run an async function synchronously (useful for REPL testing async functions).

3.3.1 async_sync

privex.helpers.asyncx.**async_sync**(*f*)

Async Synchronous Decorator, borrowed from https://stackoverflow.com/a/23036785/2648583 - added this PyDoc comment and support for returning data from a synchronous function

Allows a non-async function to run async functions using `yield from` - and can also return data

Useful for unit testing, since unittest.TestCase functions are synchronous.

Example async function:

```
>>> async def my_async_func(a, b, x=None, y=None):
...     return a, b, x, y
...
```

Using the above async function with a non-async function:

```
>>> @async_sync
... def sync_function():
...     result = yield from my_async_func(1, 2, x=3, y=4)
...     return result
...
>>> r = sync_function()
>>> print(r)
(1, 2, 3, 4,)
>>> print(r[1])
2
```

3.3.2 await_if_needed

async privex.helpers.asyncx.**await_if_needed**(*func: Union[callable, Coroutine, Awaitable, Any]*, **args*, ***kwargs*)

Call, await, and/or simply return `func` depending on whether it's an async function reference (coroutine function), a non-awaited coroutine, a standard synchronous function, or just a plain old string.

Helps take the guess work out of parameters which could be a string, a synchronous function, an async function, or a coroutine which hasn't been awaited.

```
>>> def sync_func(hello, world=1):
...     return f"sync hello: {hello} {world}"
>>> async def async_func(hello, world=1):
...     return f"async hello: {hello} {world}"
>>> await await_if_needed(sync_func, 3, world=2)
'sync hello: 3 2'
>>> await await_if_needed(async_func, 5, 4)
'async hello: 5 4'
>>> f = async_func(5, 4)
>>> await await_if_needed(f)
'async hello: 5 4'
```

Parameters

- **func** (*callable|Coroutine|Awaitable|Any*) – The function/object to await/call if needed.
- **args** – If `func` is a function/method, will forward any positional arguments to the function
- **kwargs** – If `func` is a function/method, will forward any keyword arguments to the function

Return Any func_data The result of the awaited `func`, or the original `func` if not a coroutine nor callable/awaitable

3.3.3 awaitable

`privex.helpers.asyncx.`**`awaitable`**(*func: Callable*) → Callable

Decorator which helps with creation of async wrapper functions.

Usage

Define your async function as normal, then create a standard python function using this decorator - the function should just call your async function and return it.

```
>>> async def some_func_async(a: str, b: str):
...     c = a + b
...     return c
...
>>> @awaitable
>>> def some_func(a, b) -> Union[str, Coroutine[Any, Any, str]]:
...     return some_func_async(a, b)
...
```

Now, inside of async functions, we just `await` the wrapper function as if it were the original async function.

```
>>> async def my_async_func():
...     res = await some_func("hello", "world")
...
```

While inside of synchronous functions, we call the wrapper function as if it were a normal synchronous function. The decorator will create an asyncio event loop, run the function, then return the result - transparent to the calling function.

```
>>> def my_sync_func():
...     res = some_func("hello world")
...
```

Blacklists

If you mix a lot of synchronous and asynchronous code, `sniffio` may return coroutines to synchronous functions that were called from asynchronous functions, which can of course cause problems.

To avoid this issue, you can blacklist function names, module names (and their sub-modules), and/or fully qualified module paths to functions/methods.

Three blacklists are available in this module, which allow you to specify caller functions/methods, modules, or fully qualified module paths to functions/methods for which *`awaitable()`* wrapped functions/methods should **always** execute in an event loop and return synchronously.

Example:

```
>>> from privex.helpers import asyncx
>>> # All code within the module 'some.module' and it's sub-modules will always␣
↪have awaitable's run their wrapped
>>> # functions synchronously.
>>> asyncx.AWAITABLE_BLACKLIST_MODS += ['some.module']
>>> # Whenever a function with the name 'example_func' (in any module) calls an␣
↪awaitable, it will always run synchronously
>>> asyncx.AWAITABLE_BLACKLIST_FUNCS += ['example_func']
>>> # Whenever the specific class method 'other.module.SomeClass.some_sync' calls␣
↪an awaitable, it will always run synchronously.
>>> asyncx.AWAITABLE_BLACKLIST += ['other.module.SomeClass.some_sync']
```

Original source: https://github.com/encode/httpx/issues/572#issuecomment-562179966

3.3.4 awaitable_class

privex.helpers.asyncx.**awaitable_class**(*cls: Type[T]*) → Type[T]

Wraps a class, allowing all async methods to be used in non-async code as if they were normal synchronous methods.

Example Usage

Simply decorate your class with `@awaitable_class` (no brackets! takes no arguments), and once you create an instance of your class, all of your async methods can be used by synchronous code as-if they were plain functions:

```
>>> from privex.helpers import awaitable_class
>>>
>>> @awaitable_class
>>> class ExampleAsyncCls:
>>>     async def example_async(self):
>>>         return "hello async world"
>>>
>>>     def example_sync(self):
>>>         return "hello non-async world"
>>>
```

NOTE - You can also wrap a class without using a decorator - just pass the class as the first argument like so:

```
>>> class _OtherExample:
...     async def hello(self):
...         return 'world'
>>> OtherExample = awaitable_class(_OtherExample)
```

If we call `.example_async()` on the above class from a synchronous REPL, it will return `'hello async world'` as if it were a normal synchronous method. We can also call the non-async `.example_sync()` which works like normal:

```
>>> k = ExampleAsyncCls()
>>> k.example_async()
'hello async world'
>>> k.example_sync()
'hello non-async world'
```

However, inside of an async context (e.g. an async function), `awaitable_class` will be returning coroutines, so you should `await` the methods, as you would expect when dealing with an async function:

```
>>> async def test_async():
>>>     exmp = ExampleAsyncCls()
>>>     return await exmp.example_async()
>>>
>>> await test_async()
'hello async world'
```

Parameters **cls** (*type*) – The class to wrap

Return type wrapped_class The class after being wrapped

3.3.5 call_sys_async

async privex.helpers.asyncx.**call_sys_async**(*proc*, **args*, *write: Union[bytes, str] = None*, ***kwargs*) → Tuple[bytes, bytes]

Async version of *call_sys()* - works exactly the same, other than needing to be `await`'d. Run a process `proc` with the arguments `*args`, optionally piping data (`write`) into the process's stdin - then returns the stdout and stderr of the process.

By default, `stdout` and `stdin` are set to `asyncio.PIPE` while stderr defaults to `asyncio.STDOUT`. You can override these by passing new values as keyword arguments.

While it's recommended to use the file descriptor types from the `asyncio` module, they're generally just aliases to the types in `subprocess`, meaning `subprocess.PIPE` should work the same as `asyncio.PIPE`.

Simple Example:

```
>>> from privex.helpers import call_sys_async, stringify
>>> # All arguments are automatically quoted if required, so spaces are␣
↪completely fine.
>>> folders, _ = await call_sys_async('ls', '-la', '/tmp/spaces are fine/hello␣
↪world')
>>> print(stringify(folders))
total 0
drwxr-xr-x  3 user  wheel  96  6 Dec 17:46 .
drwxr-xr-x  3 user  wheel  96  6 Dec 17:46 ..
-rw-r--r--  1 user  wheel   0  6 Dec 17:46 example
```

Piping data into a process:

```
>>> data = "hello world"
>>> # The data "hello world" will be piped into wc's stdin, and wc's stdout +␣
↪stderr will be returned
>>> out, _ = await call_sys_async('wc', '-c', write=data)
>>> int(out)
11
```

Parameters

- **proc** (*str*) – The process to execute.
- **args** (*str*) – Any arguments to pass to the process `proc` as positional arguments.
- **write** (*bytes | str*) – If this is not `None`, then this data will be piped into the process's STDIN.

Key stdout The subprocess file descriptor for stdout, e.g. `asyncio.PIPE` or `asyncio.STDOUT`

Key stderr The subprocess file descriptor for stderr, e.g. `asyncio.PIPE` or `asyncio.STDOUT`

Key stdin The subprocess file descriptor for stdin, e.g. `asyncio.PIPE` or `asyncio.STDIN`

Key cwd Set the current/working directory of the process to this path, instead of the CWD of your calling script.

Return tuple output A tuple containing the process output of stdout and stderr

3.3.6 get_async_type

`privex.helpers.asyncx.`**`get_async_type`**(*obj*) → str

Detects if `obj` is an async object/function that needs awaited / called, whether it's a synchronous callable, or whether it's unknown (probably not async)

```
>>> def sync_func(hello, world=1): return f"sync hello: {hello} {world}"
>>> async def async_func(hello, world=1): return f"async hello: {hello} {world}"
>>> get_async_type(async_func)
'coro func'
>>> get_async_type(async_func(5))
'coro'
>>> get_async_type(sync_func)
'sync func'
>>> get_async_type(sync_func(10))
'unknown'
```

Parameters **`obj`** (*Any*) – Object to check for async type

Return str async_type Either `'coro func'`, `'coro'`, `'awaitable'`, `'sync func'` or `'unknown'`

3.3.7 is_async_context

`privex.helpers.asyncx.`**`is_async_context`**() → bool

Returns `True` if currently in an async context, otherwise `False`

3.3.8 loop_run

`privex.helpers.asyncx.`**`loop_run`**(*coro: Union[Coroutine, Type[Coroutine], Callable]*, **args*, *_loop=None*, ***kwargs*) → Any

Run the coroutine or async function `coro` synchronously, using an AsyncIO event loop.

If the keyword argument `_loop` isn't specified, it defaults to the loop returned by `asyncio.get_event_loop()`

If `coro` doesn't appear to be a coroutine or async function:

- If `coro` is a normal `callable` object e.g. a function, then it'll be called.
  - If the object returned after calling `coro(*args, **kwargs)` **is a co-routine / async func**, then it'll call `loop_run` again, passing the object returned from calling it, and returning the result from that recursive call.
  - If the returned object isn't an async func / co-routine, then the object will be returned as-is.
- Otherwise, `coro` will just be returned back to the caller.

Example Usage

First we'll define the async function `some_func` to use as an example:

```
>>> async def some_func(x, y):
...     return x + y
```

Option 1 - Call an async function directly with any args/kwargs required, then pass the coroutine returned:

```
>>> loop_run(some_func(3, 4))
7
```

Option 2 - Pass a reference to the async function, and pass any required args/kwargs straight to *loop_run()* - the function will be ran with the args/kwargs you provide, then the coroutine ran in an event loop:

```
>>> loop_run(some_func, 10, y=20)     # Opt 2. Pass the async function and include␣
↪any args/kwargs for the call
30
```

Parameters

- **coro** – A co-routine, or reference to an async function to be ran synchronously
- **args** – Any positional arguments to pass to `coro` (if it's a function reference and not a coroutine)
- **_loop** (*asyncio.base_events.BaseEventLoop*) – (kwarg only!) If passed, will run `coro` in this event loop, instead of `asyncio.get_event_loop()`
- **kwargs** – Any keyword arguments to pass to `coro` (if it's a function reference and not a coroutine)

Return Any coro_result The returned data from executing the coroutine / async function

3.3.9 run_sync

`privex.helpers.asyncx.`**run_sync**(*func*, **args*, ***kwargs*)

Run an async function synchronously (useful for REPL testing async functions). (TIP: Consider using *loop_run()* instead)

> **Attention:** For most cases, you should use the function *loop_run()* instead of this. Unlike `run_sync`, *loop_run()* is able to handle async function references, coroutines, as well as coroutines / async functions which are wrapped in an outer non-async function (e.g. an `@awaitable` wrapper).
>
> *loop_run()* also supports using a custom event loop, instead of being limited to `asyncio.get_event_loop()`

Usage:

```
>>> async def my_async_func(a, b, x=None, y=None):
...     return a, b, x, y
>>>
>>> run_sync(my_async_func, 1, 2, x=3, y=4)
(1, 2, 3, 4,)
```

Parameters

- **func** (*callable*) – An asynchronous function to run
- **args** – Positional arguments to pass to `func`
- **kwargs** – Keyword arguments to pass to `func`

Classes

AwaitableMixin()	
aobject(*a, **kw)	Inheriting this class allows you to define an async __init__.

3.3.10 AwaitableMixin

class privex.helpers.asyncx.**AwaitableMixin**

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

3.3.10.1 Methods

Methods

__

3.3.11 aobject

class privex.helpers.asyncx.**aobject**(**a*, ***kw*)
Inheriting this class allows you to define an async __init__.

To use async constructors, you must construct your class using `await MyClass(params)`

Example:

```
>>> class SomeClass(aobject):
...     async def __init__(self, some_param='x'):
...         self.some_param = some_param
...         self.example = await self.test_async()
...
...     async def test_async(self):
...         return "hello world"
...
>>> async def main():
...     some_class = await SomeClass('testing')
...     print(some_class.example)
...
```

Note: Some IDEs like PyCharm may complain about having async `__new__` and `__init__`, but it **does** work with Python 3.6+.

You may be able to work-around the syntax error in your sub-class by defining your `__init__` method under a different name, and then assigning `__init__ = _your_real_init` much like this class does.

Original source: https://stackoverflow.com/a/45364670

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

3.3.11.1 Methods

Methods

`__init__`()	Initialize self.

3.3.11.1.1 __init__

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

3.4 privex.helpers.black_magic

This module contains *somewhat risky* code that uses app introspection e.g. via `inspect`.

Most functions / classes in this module will **ONLY work on CPython** (the official Python interpreter from python.org), and their functionality is not guaranteed to be stable as they interact with the interpreter to enable special functionality such as detecting the function/class/module which called your function/method.

Functions and methods in this module may be updated with breaking API changes at any time, especially if they're needed to adjust for a change in Python itself. Please ensure that any usage of this module is properly wrapped in a try/catch block, and avoid relying on functions/methods in this module for critical functionality of your application.

Most useful functions:

- *`calling_function()`* - Returns the name of the function/method which called your function/method
- *`calling_module()`* - Returns the name of the module which called your function/method
- *`caller_name()`* - Returns the fully qualified module path to the function/method/module which called your function/method

Copyright:

```
    +===================================================+
    |                 © 2019 Privex Inc.                |
    |               https://www.privex.io               |
    +===================================================+
    |                                                   |
    |        Originally Developed by Privex Inc.        |
    |        License: X11 / MIT                         |
    |                                                   |
    |        Core Developer(s):                         |
    |                                                   |
    |          (+)  Chris (@someguy123) [Privex]        |
    |          (+)  Kale (@kryogenic) [Privex]          |
    |                                                   |
    +===================================================+

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

Functions

caller_name([skip])	Get the fully qualified name of a caller in the format `some.module.SomeClass.method`
calling_function([skip])	Returns the name of the function which called your function/method.
calling_module([skip])	Returns the name of the module which called your function/method.
last_frames()	
last_stack_frame([frame_num])	

3.4.1 caller_name

`privex.helpers.black_magic.`**`caller_name`**(*skip=2*) → Optional[str]

Get the fully qualified name of a caller in the format `some.module.SomeClass.method`

> **Attention:** While class instance methods will be returned correctly, class static methods will not show up as expected. The static method `some.module.SomeClass.some_static` would be returned as `some.module.some_static`, as if it were a top-level function in the module.

Original source: https://stackoverflow.com/a/9812105

Basic Example

When used within the main program (the script you run `python3 xxx.py` on), the module will be reported as `__main__`.

File `hello.py`:

```
>>> from privex.helpers.black_magic import caller_name
>>>
>>> def f2():
...     return caller_name()
>>>
>>> def f1():
...     return f2()
...
>>> print(f"[{__name__}] f1 result: {f1()}")
[__main__] f1 result: __main__.f1
```

However, as we can see below, when we create and run `world.py` which imports `hello.py`, it correctly returns the path `hello.f1`.

File `world.py`:

```
>>> from hello import f1
>>>
>>> print(f"[{__name__}] f1 result: {f1()}")
[hello] f1 result: hello.f1
[__main__] f1 result: hello.f1
```

More Complex Example

File `some/module/hello.py`:

```
>>> from privex.helpers.black_magic import caller_name
>>>
>>> class SomeClass:
>>>     def example_method(self, skip=2):
...         return caller_name(skip)
...
```

File `some/module/world.py`:

```
>>> from some.module.hello import SomeClass
>>>
>>> class OtherClass:
...     def call_some(self, skip=2):
...         return SomeClass().example_method(skip)
...
```

File `test.py`:

```
>>> from some.module.hello import SomeClass
>>> from some.module.world import OtherClass
>>>
>>> def main_func():
...     print('SomeClass (2)', SomeClass().example_method())
...     print('OtherClass (1)', OtherClass().call_some(1))
...     print('OtherClass (2)', OtherClass().call_some())
...     print('OtherClass (3)', OtherClass().call_some(3))
...
>>> main_func()
SomeClass (2) test.main_func
OtherClass (1) some.module.hello.SomeClass.example_method
OtherClass (2) some.module.world.OtherClass.call_some
OtherClass (3) test.main_func
```

Parameters **skip** (*int*) – Specifies how many levels of stack to skip while getting caller name. `skip=1` means "who called caller_name", `skip=2` means "who called this function/method" etc.

Return str caller A fully qualified module path, e.g. `some.module.SomeClass.some_method` `None` is returned if skipped levels exceed stack height.

3.4.2 calling_function

`privex.helpers.black_magic.`**`calling_function`**(*skip=2*) → Optional[str]

Returns the name of the function which called your function/method.

Example:

```
>>> def x(skip=2): return calling_function(skip=2)
>>>
>>> def y(skip=2): return x(skip)
>>>
>>> def z(skip=2): return y(skip)
>>>
>>> print(y())    # The call to x() returns that 'y' is the function which called␣
↪it.
y
```

(continues on next page)

(continued from previous page)

```
>>> print(z())    # The call to z() calls y() -> x() - still returning that 'y' is␣
↪the caller of x()
y
>>> # If we adjust skip to 3 instead of 2, we can see that z() is the function␣
↪that called y() which called x()
>>> print(z(3))
z
```

Parameters **skip** (*int*) – Skip this many frames. 0 = calling_function() 1 = function which called calling_function() 2 = function which called the function that called calling_function() (default) and so on...

Return str|None function_name Either a string containing the function name, or `None` if you've skipped too many frames.

3.4.3 calling_module

`privex.helpers.black_magic.`**`calling_module`**(*skip=2*) → Optional[str]

Returns the name of the module which called your function/method.

Parameters **skip** (*int*) – Skip this many frames. 0 = module containing calling_function() 1 = module which called calling_function() 2 = module which called the function that called calling_function() (default) and so on...

Return str|None mod_name Either a string containing the module name, or `None` if you've skipped too many frames. If called from the main python script, then `'__main__'` will be returned instead of a proper module path.

3.4.4 last_frames

`privex.helpers.black_magic.`**`last_frames`**()

3.4.5 last_stack_frame

`privex.helpers.black_magic.`**`last_stack_frame`**(*frame_num=2*)

3.5 privex.helpers.cache

Helper functions/classes related to caching.

This module acts as a singleton wrapper, allowing for easily setting a framework-independent global cache API.

To make the module easy to use, *adapter_get()* initialises an instance of `MemoryCache` if no global cache adapter instance has been setup. This means you can use the various alias functions in this module without having to configure a cache adapter.

3.5.1 Available Cache Adapters

Standard Synchronous Adapters

Two synchronous cache adapters are included by default - `MemoryCache` (dependency free), and `RedisCache` (needs `redis` library).

While these synchronous classes don't support coroutines for most methods, as of privex-helpers 2.7 the method `privex.helpers.cache.CacheAdapter.CacheAdapter.get_or_set_async()` is an async version of `CacheAdapter.get_or_set()`, and is available on all `CacheAdapter` sub-classes (both `MemoryCache` and `RedisCache`). `get_or_set_async` allows a coroutine or coroutine function/method reference to be passed as the fallback value.

Adapter	Description
`CacheAdapter`	This is the base class for all synchronous cache adapters (doesn't do anything)
`MemoryCache`	A cache adapter which stores cached items in memory using a dict. Fully functional incl. timeout.
`RedisCache`	A cache adapter for Redis using the python library `redis`

Asynchronous (Python AsyncIO) Adapters

Over the past few years, Python's AsyncIO has grown more mature and has gotten a lot of attention. Thankfully, whether you use AsyncIO or not, we've got you covered.

Three AsyncIO cache adapters are included by default - `AsyncMemoryCache` (dependency free), `AsyncRedisCache` (needs `aioredis` library), and `AsyncMemcachedCache` (needs `aiomcache` library).

Adapter	Description
`AsyncCacheAdapter`	This is the base class for all AsyncIO cache adapters (abstract class, only implements get_or_set)
`AsyncMemoryCache`	A cache adapter which stores cached items in memory using a dict. Fully functional incl. timeout.
`AsyncRedisCache`	A cache adapter for Redis using the AsyncIO python library `aioredis`
`AsyncMemcachedCache`	A cache adapter for Memcached using the AsyncIO python library `aiomcache`

3.5.2 Setting / updating the global cache adapter instance

First import the `cache` module.

```
>>> from privex.helpers import cache
```

You must instantiate your cache adapter of choice before passing it to *`adapter_set()`* - which updates the global cache adapter instance.

```
>>> my_adapter = cache.MemoryCache()
>>> cache.adapter_set(my_adapter)
```

Once you've set the adapter, you can use the module functions such as *`get()`* and *`set()`* - or you can import `cached` to enable dictionary-like cache item access.

```
>>> cache.set('hello', 'world')
>>> cache.get('hello')
'world'
>>> from privex.helpers import cached
>>> cached['hello']
'world'
>>> cached['otherkey'] = 'testing'
```

You can also use AsyncIO adapters with the global cache adapter wrapper. *CacheWrapper* uses *awaitable()* to ensure that AsyncIO adapters can work synchronously when being called from a synchronous function, while working asynchronously from a non-async function.

```
>>> my_adapter = cache.AsyncRedisCache()
>>> cache.adapter_set(my_adapter)
>>>
>>> # get_hello_async() is async, so @awaitable returns the normal .get() coroutine␣
↪for awaiting
>>> async def get_hello_async():
...     result = await cached.get('hello')
...     return result
...
>>> # get_hello() is synchronous, so @awaitable seamlessly runs .get() in an event␣
↪loop and returns
>>> # the result - get_hello() can treat it as if it were just a normal synchronous␣
↪function.
>>> def get_hello():
...     return cached.get('hello')
...
>>> get_hello()
'world'
>>> await get_hello_async()
'world'
```

3.5.3 Plug-n-play usage

As explained near the start of this module's documentation, you don't have to set the global adapter if you only plan on using the simple `MemoryCache` adapter.

Just start using the global cache API via either *privex.helpers.cache* or `privex.helpers.cache.cached` and MemoryCache will automatically be instantiated as the global adapter as soon as something attempts to access the global instance.

We recommend importing `cached` rather than `cache`, as it acts as a wrapper that allows dictionary-like cache key getting/setting, and is also immediately aware when the global cache adapter is set/replaced.

```
>>> from privex.helpers import cached
```

You can access `cached` like a dictionary to get and set cache keys (they will use the default expiry time of `privex.helpers.settings.DEFAULT_CACHE_TIMEOUT`)

```
>>> cached['testing'] = 123
>>> cached['testing']
123
```

You can also call methods such as *get()* and *set()* for getting/setting cache items with more control, for example:

1. Setting a custom expiration, or disabling expiration by setting timeout to `None`

```
>>> cached.set('example', 'test', timeout=30)    # Drop 'example' from the cache␣
↪after 30 seconds from now.
>>> cached.set('this key', 'is forever!', timeout=None) # A timeout of ``None``␣
↪disables automatic expiration.
```

2. Fallback values when a key isn't found, or have it throw an exception if it's not found instead.

```
>>> cached.get('example', 'NOT FOUND')          # If the key 'example' doesn't␣
↪exist, return 'NOT FOUND'
'test'
```

```
>>> try:   # By setting ``fail`` to True, ``get`` raises ``CacheNotFound`` if the␣
↪key doesn't exist / is expired
...     cached.get('nonexistent', fail=True)
... except CacheNotFound:
...     log.error('The cache key "nonexistent" does not exist!')
>>>
```

3. Using *get_or_set()* you can specify either a standard type (e.g. `str`, `int`, `dict`), or even a custom function to call to obtain the value to set and return.

```
>>> cached.get_or_set('hello', lambda key: 'world', timeout=60)
>>> cached['hello']
'world'
```

Copyright:

```
    +==================================================+
    |                © 2019 Privex Inc.                |
    |              https://www.privex.io               |
    +==================================================+
    |                                                  |
    |        Originally Developed by Privex Inc.       |
    |        License: X11 / MIT                        |
    |                                                  |
    |        Core Developer(s):                        |
    |                                                  |
    |          (+)  Chris (@someguy123) [Privex]       |
    |          (+)  Kale (@kryogenic) [Privex]         |
    |                                                  |
    +==================================================+

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

Attributes

—

Functions

adapter_get([default])	Get the global cache adapter instance.
adapter_set(adapter)	Set the global cache adapter instance to `adapter` - which should be an instantiated adapter class which implements `CacheAdapter`
get(key[, default, fail])	Return the value of cache key `key`.
get_or_set(key, value[, timeout])	Attempt to return the value of `key` in the cache.
remove(*key)	Remove one or more keys from the cache.
set(key, value[, timeout])	Set the cache key `key` to the value `value`, and automatically expire the key after `timeout` seconds from now.
update_timeout(key[, timeout])	Update the timeout for a given `key` to `datetime.utcnow() + timedelta(seconds=timeout)`

3.5.3.1 adapter_get

privex.helpers.cache.**adapter_get**(*default: Type[privex.helpers.cache.CacheAdapter.CacheAdapter] = <class 'privex.helpers.cache.MemoryCache.MemoryCache'>*) → privex.helpers.cache.CacheAdapter.CacheAdapter

Get the global cache adapter instance. If there isn't one, then by default this function will initialise `MemoryAdapter` and set it as the global cache adapter.

To set the global cache adapter instance, use *adapter_set()*

To use a different fallback class, pass a class name which implements `CacheAdapter` like so:

```
>>> adapter_get(default=MemoryCache)
```

Parameters **default** –

Returns

3.5.3.2 adapter_set

privex.helpers.cache.**adapter_set**(*adapter: privex.helpers.cache.CacheAdapter.CacheAdapter*)

Set the global cache adapter instance to `adapter` - which should be an instantiated adapter class which implements `CacheAdapter`

Example:

```
>>> from privex.helpers import cache
>>> cache.adapter_set(cache.MemoryCache())
```

Parameters **adapter** (*CacheAdapter*) – An instance of a class which implements `CacheAdapter` for global use.

Return CacheAdapter adapter A reference to your adapter from `__STORE['adapter']`

3.5.3.3 get

privex.helpers.cache.**get**(*key: str*, *default: Any = None*, *fail: bool = False*) → Any

Return the value of cache key `key`. If the key wasn't found, or it was expired, then `default` will be returned.

Optionally, you may choose to pass `fail=True`, which will cause this method to raise `CacheNotFound` instead of returning `default` when a key is non-existent / expired.

Parameters

- **key** (*str*) – The cache key (as a string) to get the value for, e.g. `example:test`
- **default** (*Any*) – If the cache key `key` isn't found / is expired, return this value (Default: `None`)
- **fail** (*bool*) – If set to `True`, will raise `CacheNotFound` instead of returning `default` when a key is non-existent / expired.

Raises **CacheNotFound** – Raised when `fail=True` and `key` was not found in cache / expired.

Return Any value The value of the cache key `key`, or `default` if it wasn't found.

3.5.3.4 get_or_set

privex.helpers.cache.**get_or_set**(*key: str*, *value: Union[Any, callable]*, *timeout: int = 300*) → Any

Attempt to return the value of `key` in the cache. If `key` doesn't exist or is expired, then it will be set to `value`, and `value` will be returned.

The `value` parameter can be any standard type such as `str` or `dict` - or it can be a callable function / method which returns the value to set and return.

Basic Usage:

```
>>> from privex.helpers import cache as c
>>> c.get('testing')
None
>>> c.get_or_set('testing', 'hello world')
'hello world'
>>> c.get('testing')
'hello world'
```

Set and get the value from a function if ``key`` didn't exist / was expired:

```
>>> def my_func(): return "hello world"
>>> c.get_or_set('example', my_func)
'hello world'
>>> c.get('example')
'hello world'
```

Parameters

- **key** (*str*) – The cache key (as a string) to get/set the value for, e.g. `example:test`
- **value** (*Any*) – The value to store in the cache key `key`. Can be a standard type, or a callable function.
- **timeout** (*int*) – The amount of seconds to keep the data in cache. Pass `None` to disable expiration.

Return Any value The value of the cache key `key`, or `value` if it wasn't found.

3.5.3.5 remove

privex.helpers.cache.**remove**(**key: str*) → bool

Remove one or more keys from the cache.

If all cache keys existed before removal, `True` will be returned. If some didn't exist (and thus couldn't remove), then `False` will be returned.

Parameters **key** (*str*) – The cache key(s) to remove

Return bool removed `True` if `key` existed and was removed

Return bool removed `False` if `key` didn't exist, and no action was taken.

3.5.3.6 set

privex.helpers.cache.**set**(*key: str*, *value: Any*, *timeout: Optional[int] = 300*)

Set the cache key `key` to the value `value`, and automatically expire the key after `timeout` seconds from now.

If `timeout` is `None`, then the key will never expire (unless the cache implementation loses it's persistence, e.g. memory caches with no disk writes).

Parameters

- **key** (*str*) – The cache key (as a string) to set the value for, e.g. `example:test`
- **value** (*Any*) – The value to store in the cache key `key`
- **timeout** (*int*) – The amount of seconds to keep the data in cache. Pass `None` to disable expiration.

3.5.3.7 update_timeout

privex.helpers.cache.**update_timeout**(*key: str*, *timeout: int = 300*) → Any

Update the timeout for a given `key` to `datetime.utcnow() + timedelta(seconds=timeout)`

This method allows keys which are already expired, allowing expired cache keys to have their timeout extended **after** expiry.

Example:

```
>>> from privex.helpers import cache
>>> from time import sleep
>>> cache.set('example', 'test', timeout=60)
>>> sleep(70)
>>> cache.update_timeout('example', timeout=60)    # Reset the timeout for ``
↪'example'`` to ``now + 60 seconds``
>>> cache.get('example')
'test'
```

Parameters

- **key** (*str*) – The cache key to update the timeout for
- **timeout** (*int*) – Reset the timeout to this many seconds from `datetime.utcnow()`

Raises **CacheNotFound** – Raised when `key` was not found in cache (thus cannot extend timeout)

Return Any value The value of the cache key

Classes

CacheWrapper()	**CacheWrapper** is a small class designed to wrap an instance of `CacheAdapter` and allow the adapter to be switched out at any time, using the static class attribute *cache_instance*.

3.5.3.8 CacheWrapper

class privex.helpers.cache.**CacheWrapper**

CacheWrapper is a small class designed to wrap an instance of `CacheAdapter` and allow the adapter to be switched out at any time, using the static class attribute *cache_instance*.

This class is used for the singleton global variable `cached`

For convenience, if *cache_instance* isn't set-up when something makes an adapter-dependant call, then the adapter class in `default_adapter` will be instantiated and stored in `cache_instance`

```
>>> # Using the ``: CacheAdapter`` type hinting will allow most IDEs to treat the␣
↪wrapper as if it were
>>> # a normal CacheAdapter child class, thus showing appropriate completion /␣
↪usage warnings
>>> c: CacheAdapter = CacheWrapper()
>>> c.set('hello', 'world')
>>> c['hello']
'world'
```

You can replace the cache adapter singleton using the module function *adapter_set()* (recommended)

```
>>> from privex.helpers import cache, CacheWrapper
>>> cache.adapter_set(cache.MemoryCache())    # Set the current adapter for both␣
↪the cache module, and wrapper.
```

If you only plan to use this wrapper, then you can use *set_adapter()* to update the current cache adapter instance.

```
>>> CacheWrapper.set_adapter(cache.MemoryCache())  # Set the adapter only for the␣
↪wrapper (aka ``cached``)
```

__init__()

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

3.5.3.8.1 Methods

Methods

get_adapter([default])	Attempt to get the singleton cache adapter from *cache_instance* - if the instance is `None`, then attempt to instantiate `default()`
set_adapter(adapter)	

3.5.3.8.1.1 get_adapter

static CacheWrapper.**get_adapter**(*default: Type[privex.helpers.cache.CacheAdapter.CacheAdapter] = <class 'privex.helpers.cache.MemoryCache.MemoryCache'>*, **args*, ***kwargs*) → privex.helpers.cache.CacheAdapter.CacheAdapter

Attempt to get the singleton cache adapter from *cache_instance* - if the instance is None, then attempt to instantiate default()

If any *args or **kwargs are passed, they will be passed through to default(*args, **kwargs) so that any necessary configuration parameters can be passed to the class.

3.5.3.8.1.2 set_adapter

static CacheWrapper.**set_adapter**(*adapter: privex.helpers.cache.CacheAdapter.CacheAdapter*) → privex.helpers.cache.CacheAdapter.CacheAdapter

3.5.3.8.2 Attributes

Attributes

cache_instance	Holds the singleton instance of a CacheAdapter implementation

3.5.3.8.2.1 cache_instance

CacheWrapper.**cache_instance: privex.helpers.cache.CacheAdapter.CacheAdapter = None**

Holds the singleton instance of a CacheAdapter implementation

3.6 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.        |
    |        License: X11 / MIT                         |
    |                                                   |
    |        Core Developer(s):                         |
    |                                                   |
    |          (+)  Chris (@someguy123) [Privex]        |
    |          (+)  Kale (@kryogenic) [Privex]          |
    |                                                   |
    +===================================================+

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

Attributes

—

Functions

almost(compare, *numbers[, tolerance])	Compare two or more numbers, returning `True` if all `numbers` are no more than `tolerance` greater or smaller than than `compare` - otherwise `False`.
byteify(data[, encoding, if_none])	Convert a piece of data into bytes if it isn't already.
call_sys(proc, *args[, write])	A small wrapper around `subprocess.Popen` which allows executing processes, while optionally piping data (`write`) into the process's stdin, then finally returning the process's output and error results.
camel_to_snake(name)	Convert `name` from camel case (`HelloWorld`) to snake case (`hello_world`).
chunked(iterable, n)	Split iterable into `n` iterables of similar size
construct_dict(cls, kwargs[, args, ...])	Removes keys from the passed dict `data` which don't exist on `cls` (thus would get rejected as kwargs) using *get_function_params()*.
dec_round(amount[, dp, rounding])	Round a Decimal to x decimal places using `quantize` (`dp` must be >= 1 and the default dp is 2)
empty(v[, zero, itr])	Quickly check if a variable is empty or not.
empty_if(v[, is_empty, not_empty])	Syntactic sugar for `x if empty(y) else z`.
env_bool(env_key[, env_default])	Obtains an environment variable `env_key`, if it's empty or not set, `env_default` will be returned.
env_cast(env_key, cast[, env_default])	Obtains an environment variable `env_key`, if it's empty or not set, `env_default` will be returned.
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_decimal(env_key[, env_default])	Alias for *env_cast()* with `Decimal` casting
env_int(env_key[, env_default])	Alias for *env_cast()* with `int` casting
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)]
extract_settings(prefix[, _settings, ...])	Extract prefixed settings from a given module, dictionary, class, or instance.
filter_form(form, *keys[, cast])	Extract the keys `keys` from the dict-like `form` if they exist and return a dictionary containing the keys and values found.
get_function_params(obj[, check_parents])	Extracts a function/method's signature (or class constructor signature if a class is passed), and returns it as a dictionary.
human_name(class_name)	This function converts a class/function name into a Title Case name.
inject_items(items, dest_list, position)	Inject a list `items` after a certain element in `dest_list`.
io_tail(f[, nlines, bsz])	NOTE: If you're only loading a small amount of lines, e.g. less than 1MB, consider using the much easier *tail()*

continues on next page

Table 14 – continued from previous page

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).
reverse_io(f[, blocksize])	Read file as series of blocks from end of file to start.
shell_quote(*args)	Takes command line arguments as positional args, and properly quotes each argument to make it safe to pass on the command line.
stringify(data[, encoding, if_none])	Convert a piece of data into a string (from bytes) if it isn't already.
tail(filename[, nlines, bsz])	Pure python equivalent of the UNIX `tail` command.

3.6.1 almost

privex.helpers.common.**almost**(*compare: Union[decimal.Decimal, int, float, str]*, **numbers: Union[decimal.Decimal, int, float, str]*, *tolerance: Union[decimal.Decimal, int, float, str] = Decimal('0.01')*, ***kwargs*) → bool

Compare two or more numbers, returning `True` if all `numbers` are no more than `tolerance` greater or smaller than than `compare` - otherwise `False`.

Works similarly to `unittest.TestCase.assertAlmostEqual()`

Basic usage with two numbers + default tolerance (`0.01`):

```
>>> almost('5', '5.001')
True
>>> almost('5', '5.5')
False
```

Multiple numbers + custom tolerance:

```
>>> almost('5', '5.14', '4.85', '5.08', tolerance=Decimal('0.2'))
True
>>> almost('5', '5.3', '4.85', '5.08', tolerance=Decimal('0.2'))
False
```

Using `fail` or `test`:

```
>>> # By passing ``fail=True``, a descriptive AssertionError is raised when the␣
↪tolerance check fails.
>>> almost('5', '5.01', fail=True)
True
>>> almost('5', '5.02', fail=True)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "privex/helpers/common.py", line 1044, in almost
```

(continues on next page)

(continued from previous page)

```
    raise AssertionError(
AssertionError: Number at position 0 (val: 5.02) failed tolerance (0.01) check␣
↪against 5
>>> # By passing ``test=True``, a standard ``assert`` will be used to compare the␣
↪numbers.
>>> almost('5', '5.01', test=True)
True
>>> almost('5', '5.02', test=True)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "privex/helpers/common.py", line 1041, in almost
    assert (x - tolerance) <= compare <= (x + tolerance)
AssertionError
```

Parameters

- **compare** (*Decimal | int | float*) – The base number which all `numbers` will be compared against.
- **numbers** (*Decimal | int | float*) – One or more numbers to compare against `compare`
- **tolerance** (*Decimal | int | float*) – (kwarg only) Amount that each `numbers` can be greater/smaller than `compare` before returning `False`.
- **fail** (*bool*) – (default: `False`) If true, will raise `AssertionError` on failed tolerance check, instead of returning `False`. (mutually exclusive with `assert`)
- **test** (*bool*) – (default: `False`) If true, will use `assert` instead of testing with `if`. Useful in unit tests. (mutually exclusive with `raise`)

Raises

- **AttributeError** – When less than 1 number is present in `numbers`
- **AssertionError** – When kwarg `raise` is `True` and one or more numbers failed the tolerance check.

Return bool is_almost `True` if all `numbers` are within `tolerance` of `compare`, `False` if one or more `numbers` is outside of the tolerance.

3.6.2 byteify

`privex.helpers.common.`**`byteify`**(*data: Optional[Union[str, bytes]]*, *encoding='utf-8'*, *if_none=None*) → bytes

Convert a piece of data into bytes if it isn't already:

```
>>> byteify("hello world")
b"hello world"
```

By default, if `data` is `None`, then a `TypeError` will be raised by `bytes()`.

If you'd rather convert `None` into a blank bytes string, use `if_node=""`, like so:

```
>>> byteify(None)
TypeError: encoding without a string argument
>>> byteify(None, if_none="")
b''
```

3.6.3 call_sys

privex.helpers.common.**call_sys**(*proc*, **args*, *write: Union[bytes, str] = None*, ***kwargs*) → Tuple[bytes, bytes]

A small wrapper around `subprocess.Popen` which allows executing processes, while optionally piping data (`write`) into the process's stdin, then finally returning the process's output and error results. Designed to be easier to use than using `subprocess.Popen` directly.

Using AsyncIO? - there's a native python asyncio version of this function available in *call_sys_async()*, which uses the native `asyncio.subprocess.create_subprocess_shell()`, avoiding blocking IO.

By default, `stdout` and `stdin` are set to `subprocess.PIPE` while stderr defaults to `subprocess.STDOUT`. You can override these by passing new values as keyword arguments.

NOTE: The first positional argument is executed, and all other positional arguments are passed to the process in the order specified. To use call_sys's arguments `write`, `stdout`, `stderr` and/or `stdin`, you **MUST** specify them as keyword arguments, otherwise they'll just be passed to the process you're executing.

Any keyword arguments not specified in the `:param` or `:key` pydoc specifications will simply be forwarded to the `subprocess.Popen` constructor.

Simple Example:

```
>>> # All arguments are automatically quoted if required, so spaces are␣
↪completely fine.
>>> folders, _ = call_sys('ls', '-la', '/tmp/spaces are fine/hello world')
>>> print(stringify(folders))
backups  cache   lib  local  lock  log  mail  opt  run  snap  spool  tmp
```

Piping data into a process:

```
>>> data = "hello world"
>>> # The data "hello world" will be piped into wc's stdin, and wc's stdout +␣
↪stderr will be returned
>>> out, _ = call_sys('wc', '-c', write=data)
>>> int(out)
11
```

Parameters

- **proc** (*str*) – The process to execute.
- **args** (*str*) – Any arguments to pass to the process `proc` as positional arguments.
- **write** (*bytes | str*) – If this is not `None`, then this data will be piped into the process's STDIN.

Key stdout The subprocess file descriptor for stdout, e.g. `subprocess.PIPE` or `subprocess.STDOUT`

Key stderr The subprocess file descriptor for stderr, e.g. `subprocess.PIPE` or `subprocess.STDOUT`

Key stdin The subprocess file descriptor for stdin, e.g. `subprocess.PIPE` or `subprocess.STDIN`

Key cwd Set the current/working directory of the process to this path, instead of the CWD of your calling script.

Return tuple output A tuple containing the process output of stdout and stderr

3.6.4 camel_to_snake

privex.helpers.common.**camel_to_snake**(*name: Union[bytes, str]*) → str

Convert `name` from camel case (`HelloWorld`) to snake case (`hello_world`).

`name` can be either a `str` or `bytes`.

Example:

```
>>> camel_to_snake("HelloWorldLoremIpsum")
'hello_world_lorem_ipsum'
```

Parameters **name** (*str|bytes*) – A camel case (class style) name, e.g. `HelloWorld`

Return str snake_case `name` converted to snake case `hello_world`

3.6.5 chunked

privex.helpers.common.**chunked**(*iterable*, *n*)

Split iterable into `n` iterables of similar size

Examples::

```
>>> l = [1, 2, 3, 4]
>>> list(chunked(l, 4))
[[1], [2], [3], [4]]
```

```
>>> l = [1, 2, 3]
>>> list(chunked(l, 4))
[[1], [2], [3], []]
```

```
>>> l = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> list(chunked(l, 4))
[[1, 2, 3], [4, 5, 6], [7, 8, 9], [10]]
```

Taken from: https://stackoverflow.com/a/24484181/2648583

3.6.6 construct_dict

privex.helpers.common.**construct_dict**(*cls: Union[Type[T], C]*, *kwargs: dict*, *args: Iterable = None*, *check_parents=True*) → Union[T, Any]

Removes keys from the passed dict `data` which don't exist on `cls` (thus would get rejected as kwargs) using *`get_function_params()`*. Then create and return an instance of `cls`, passing the filtered `kwargs` dictionary as keyword args.

Ensures that any keys in your dictionary which don't exist on `cls` are automatically filtered out, instead of causing an error due to unexpected keyword arguments.

Example - User class which only takes specific arguments

First let's define a class which only takes three arguments in it's constructor - username, first_name, last_name.

```
>>> class User:
...     def __init__(self, username, first_name=None, last_name=None):
...         self.username = username
```

(continues on next page)

(continued from previous page)

```
...         self.first_name, self.last_name = first_name, last_name
...
```

Now we'll create a dictionary which has those three arguments, but also the excess `address` and `phone`.

```
>>> data = dict(username='johndoe123', first_name='John', last_name='Doe',
...             address='123 Example St', phone='+1-123-000-1234')
```

If we tried to directly pass data as keyword args, we'd get an error:

```
>>> john = User(**data)
TypeError: __init__() got an unexpected keyword argument 'address'
```

But by using *construct_dict()*, we're able to construct a `User`, as this helper function detects that the excess `address` and `phone` are not valid parameters for `User`'s constructor.

```
>>> from privex.helpers import construct_dict
>>> john = construct_dict(User, data)
>>> print(john.username, john.first_name, john.last_name)
johndoe123 John Doe
```

Example - A function/method which only takes specific arguments

Not only can *construct_dict()* be used for classes, but it can also be used for any function/method.

Here's an example using a simple "factory function" which creates user objects:

```
>>> def create_user(username, first_name=None, last_name=None):
...     return User(username, first_name, last_name)
>>>
>>> data = dict(
...     username='johndoe123', first_name='John', last_name='Doe',
...     address='123 Example St', phone='+1-123-000-1234'
... )
>>> # We can't just pass data as kwargs due to the extra keys.
>>> create_user(**data)
TypeError: create_user() got an unexpected keyword argument 'address'
>>> # But we can call the function using construct_dict, which filters out the␣
↪excess dict keys :)
>>> john = construct_dict(create_user, data)
>>> print(john.username, john.first_name, john.last_name)
johndoe123 John Doe
```

Parameters

- **cls** (*Type[T] | callable*) – A class (not an instance) or callable (function / lambda) to extract and filter the parameter's from, then call using filtered `kwargs` and `args`.
- **kwargs** (*dict*) – A dictionary containing keyword arguments to filter and use to call / construct `cls`.
- **args** (*list | set*) – A list of positional arguments (NOT FILTERED!) to pass when calling/constructing `cls`.
- **check_parents** (*bool*) – (Default: `True`) If `obj` is a class and this is True, will recursively grab the constructor parameters for all parent classes of `cls` and merge them into the returned dict.

Return Any func_result If `cls` was a function/method, the return result will be the returned data/object from the function passed.

Return T cls_instance If `cls` was a class, then the return result will be an instance of the class.

3.6.7 dec_round

privex.helpers.common.**dec_round**(*amount: decimal.Decimal*, *dp: int = 2*, *rounding=None*) → decimal.Decimal

Round a Decimal to x decimal places using `quantize` (`dp` must be >= 1 and the default dp is 2)

If you don't specify a rounding option, it will use whatever rounding has been set in `decimal.getcontext()` (most python versions have this default to `ROUND_HALF_EVEN`)

Basic Usage:

```
>>> from decimal import Decimal, getcontext, ROUND_FLOOR
>>> x = Decimal('1.9998')
>>> dec_round(x, 3)
Decimal('2.000')
```

Custom Rounding as an argument:

```
>>> dec_round(x, 3, rounding=ROUND_FLOOR)
Decimal('1.999')
```

Override context rounding to set the default:

```
>>> getcontext().rounding = ROUND_FLOOR
>>> dec_round(x, 3)
Decimal('1.999')
```

Parameters

- **amount** (*Decimal*) – The amount (as a Decimal) to round
- **dp** (*int*) – Number of decimal places to round `amount` to. (Default: 2)
- **rounding** (*str*) – A `decimal` rounding option, e.g. `ROUND_HALF_EVEN` or `ROUND_FLOOR`

Return Decimal rounded The rounded Decimal amount

3.6.8 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')
...
```

(continues on next page)

(continued from previous page)

```
>>> 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)

3.6.9 empty_if

privex.helpers.common.**empty_if**(*v: V*, *is_empty: K = None*, *not_empty: T = <class 'privex.helpers.types.UseOrigVar'>*, ***kwargs*) → Union[T, K, V]

Syntactic sugar for `x if empty(y) else z`. If `not_empty` isn't specified, then the original value `v` will be returned if it's not empty.

Example 1:

```
>>> def some_func(name=None):
...     name = empty_if(name, 'John Doe')
...     return name
>>> some_func("")
John Doe
>>> some_func("Dave")
Dave
```

Example 2:

```
>>> empty_if(None, 'is empty', 'is not empty')
is empty
>>> empty_if(12345, 'is empty', 'is not empty')
is not empty
```

Parameters

- **v** (*Any*) – The value to test for emptiness
- **is_empty** – The value to return if `v` is empty (defaults to `None`)
- **not_empty** – The value to return if `v` is not empty (defaults to the original value `v`)
- **kwargs** – Any additional kwargs to pass to *empty()*

Key zero if `zero=True`, then v is empty if it's int `0` or str `'0'`

Key itr if `itr=True`, then v is empty if it's `[]`, `{}`, or is an iterable and has 0 length

Return V orig_var The original value `v` is returned if `not_empty` isn't specified.

Return K is_empty The value specified as `is_empty` is returned if `v` is empty

Return T not_empty The value specified as `not_empty` is returned if `v` is not empty (and not_empty was specified)

3.6.10 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)

3.6.11 env_cast

`privex.helpers.common.`**`env_cast`**(*env_key: str*, *cast: callable*, *env_default=None*)

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 type of your choice using the callable `cast` parameter

Example:

```
>>> os.environ['HELLO'] = '1.234'
>>> env_cast('HELLO', Decimal, Decimal('0'))
Decimal('1.234')
```

Parameters

- **`cast`** (*`callable`*) – A function to cast the user's env data such as `int` `str` or `Decimal` etc.
- **`env_key`** (*`str`*) – Environment var to attempt to load
- **`env_default`** (*`any`*) – Fallback value if the env var is empty / not set (Default: None)

3.6.12 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

3.6.13 env_decimal

privex.helpers.common.**env_decimal**(*env_key: str*, *env_default=None*) → decimal.Decimal

Alias for *env_cast()* with `Decimal` casting

3.6.14 env_int

privex.helpers.common.**env_int**(*env_key: str*, *env_default=None*) → int

Alias for *env_cast()* with `int` casting

3.6.15 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 `,`)

3.6.16 extract_settings

privex.helpers.common.**extract_settings**(*prefix: str*, *_settings=<module 'privex.helpers.settings' from '/home/docs/checkouts/readthedocs.org/user_builds/python-helpers/checkouts/develop/privex/helpers/settings.py'>*, *defaults=None*, *merge_conf=None*, ***kwargs*) → dict

Extract prefixed settings from a given module, dictionary, class, or instance.

This helper function searches the object `_settings` for keys starting with `prefix`, and for any matching keys, it removes the prefix from each key, converts the remaining portion of each key to lowercase (unless you've set `_case_sensitive=True`), and then returns the keys their linked values as a `dict`.

For example, if you had a file called `myapp/settings.py` which contained `REDIS_HOST = 'localhost'` and `REDIS_PORT = 6379`, you could then run:

```
>>> # noinspection PyUnresolvedReferences
>>> from myapp import settings
>>> extract_settings('REDIS_', settings)
{'host': 'localhost', 'port': 6379}
```

Example settings module at `myapp/settings.py`

```
from os.path import dirname, abspath, join

BASE_DIR = dirname(dirname(dirname(abspath(__file__))))
VERSION_FILE = join(BASE_DIR, 'privex', 'helpers', '__init__.py')

REDIS_HOST = 'localhost'
REDIS_PORT = 6379
REDIS_DB = 0

DEFAULT_CACHE_TIMEOUT = 300
```

Example - Extract Redis settings:

```
>>> # noinspection PyUnresolvedReferences
>>> from myapp import settings
>>> from privex.helpers import extract_settings
>>>
>>> # All keyword arguments (apart from _settings_mod and _keys_lower) are␣
↪converted into a dictionary
>>> # and merged with the extracted settings
>>> # noinspection PyTypeChecker
>>> extract_settings('REDIS_', _settings=settings, port=6479, debug=True)
{'host': 'localhost', 'port': 6379, 'db': 0, 'debug': True}
>>> extract_settings('REDIS_', _settings=settings, merge_conf=dict(port=6479))
{'host': 'localhost', 'port': 6479, 'db': 0}
```

Example - Extract Redis settings - case sensitive mode:

```
>>> extract_settings('REDIS_', _settings=settings, _case_sensitive=True)
{'HOST': 'localhost', 'PORT': 6379, 'DB': 0}
```

Example - Extract database settings from the environment

The below dict comprehension is just so you can see the original environment keys before we run `extract_settings`:

```
>>> import os
>>> from privex.helpers import extract_settings
>>>
>>> {k: v for k,v in os.environ.items() if 'DB_' in k}
{'DB_USER': 'root',
 'DB_PASS': 'ExamplePass',
 'DB_NAME': 'example_db'}
```

We'll now call `extract_settings` using `os.environ` converted into a dictionary, and attempt to quickly obtain the database settings - with lowercase keys, and without their `DB_` prefix.

Below, you'll see extract_settings extracted all keys starting with **DB_**, removed the **DB_** prefix, converted the remaining portion of the key to lowercase, and also merged in the default setting 'host' since DB_HOST didn't exist.

The outputted dictionary is perfect for passing to many database library constructors:

```
>>> extract_settings('DB_', dict(os.environ), host='localhost')
{'user': 'root',
 'pass': 'ExamplePass',
 'name': 'example_db',
 'host': 'localhost'}
```

Parameters

- **prefix** (*str*) – The prefix (including the first underscore (_) or other separator) to search for in the settings
- **_settings** (*Module|dict|object*) – The object to extract the settings from. The object can be one of the following:
  - A `module`, for example passing `settings` after running `from myapp import settings`
  - A `dict`, for example `extract_settings('X_', dict(X_A=1, X_B=2))`
  - A class which has the desired settings defined on it's `.__dict__` (e.g. any standard user class - `class MyClass:`, with settings defined as static class attributes)
  - An instance of a class, which has all desired settings defined inside of `.__dict__` (e.g. any standard user class instance, with static and/or instance attributes for each setting)
  - Any other type which supports being casted to a dictionary via `dict(obj)`.
- **merge_conf** (*dict*) – Optionally you may specify a dictionary of "override" settings to merge with the extracted settings. The values in this dictionary take priority over both `defaults`, and the keys from `_settings`.
- **defaults** (*dict*) – Optionally you may specify a dictionary of default settings to merge **before** the extracted settings, meaning values are only used if the key wasn't present in the extracted settings nor `merge_conf`.
- **kwargs** – Additional settings as keyword arguments (see below). Any keyword argument keys which aren't valid settings will be added to the `defaults` dictionary. This means that defaults can also be specified as kwargs - as long as they don't clash with any used kwarg settings (see below).

Key _case_sensitive (Default `False`) If `True`, `prefix` is compared against `_settings` keys case sensitively. If `False`, then both `prefix` and each `_settings` key is converted to lowercase before comparison.

Key _keys_lower Defaults to `True` if _case_sensitive is False, and `False` if _case_sensitive is True. If `True`, each extracted settings key is converted to lowercase before returning them - otherwise they're returned with the same case as they were in `_settings`.

Return dict config The extracted configuration keys (without their prefixes) and values as a dictionary. Based on the extracted keys from `_settings`, the fallback settings in `defaults` (and excess `kwargs`), plus the override settings in `merge_conf`.

3.6.17 filter_form

`privex.helpers.common.`**`filter_form`**(*form: Mapping*, **keys*, *cast: callable = None*) → Dict[str, Any]

Extract the keys `keys` from the dict-like `form` if they exist and return a dictionary containing the keys and values found.

Optionally, if `cast` isn't `None`, then `cast` will be called to cast each `form` value to the desired type, e.g. `int`, `Decimal`, or `str`.

Example usage:

```
>>> a = dict(a=1, c=2, d=3)
>>> filter_form(a, 'a', 'c', 'e')
{'a': 1, 'c': 2}
>>> b = dict(lorem=1, ipsum='2', dolor=5.67)
>>> filter_form(b, 'lorem', 'ipsum', 'dolor', cast=int)
{'lorem': 1, 'ipsum': 2, 'dolor': 5}
```

Parameters

- **`form`** (*`Mapping`*) – A dict-like object to extract `key` from.
- **`keys`** (*`str|Any`*) – One or more keys to extract from `form`
- **`cast`** (*`callable`*) – Cast the value of any extract `form` key using this callable

Return dict filtered_form A dict containing the extracted keys and respective values from `form`

3.6.18 get_function_params

`privex.helpers.common.`**`get_function_params`**(*obj: Union[type, callable]*, *check_parents=False*, ***kwargs*) → Union[Dict[str, inspect.Parameter], *privex.helpers.collections.DictObject*, Dict[type, Dict[str, inspect.Parameter]]]

Extracts a function/method's signature (or class constructor signature if a class is passed), and returns it as a dictionary.

Primarily used by *`construct_dict()`* - but may be useful for other purposes.

If you've passed a class, you can set `check_parents` to `True` to obtain the signatures of the passed class's constructor AND all of it's parent classes, returned as a dictionary mapping classes to dictionaries of parameters.

If you've set `check_parents` to `True`, but you want the parameters to be a flat dictionary (just like when passing a function or class without check_parents), you can also pass `merge=True`, which merges each class's constructor parameters into a dictionary mapping names to `inspect.Parameter` objects.

If any parameters conflict, children's constructor parameters always take precedence over their parent's version, much in the same way that Python's inheritance works.

Basic (with functions):

```
>>> def some_func(x, y, z=123, *args, **kwargs):
...     pass
```

Get all normal parameters (positional and kwargs - excluding catch-all `*args` / `**kwargs` parameter types):

```
>>> params = get_function_params(some_func)
>>> params
{'x': <Parameter "x">, 'y': <Parameter "y">, 'z': <Parameter "z=123">}
```

Get raw parameter name and value (as written in signature) / access default values:

```
>>> str(params.z.name)      # You can also access it via params['z']
'z=123'
>>> params.z.default  # You can also access it via params['z']
123
```

Get only **required** parameters:

```
>>> get_function_params(some_func, ignore_defaults=True)
{'x': <Parameter "x">, 'y': <Parameter "y">}
```

Get only parameters with defaults:

```
>>> get_function_params(some_func, ignore_positional=True)
{'z': <Parameter "z=123">}
```

Example Usage (with classes and sub-classes):

```
>>> class BaseClass:
...     def __init__(self, a, b, c=1234, **kwargs):
...         pass

>>> class Example(BaseClass):
...     def __init__(self, d, e='hello', f=None, a='overridden', **kwargs):
...         super().__init__(a=a, d=d, e=e, f=f, **kwargs)
```

If we pass the class `Example` on it's own, we get a dictionary of just it's own parameters:

```
>>> get_function_params(Example)
{'d': <Parameter "d">, 'e': <Parameter "e='hello'">, 'f': <Parameter "f=None">}
```

However, if we set `check_parents=True`, we now get a dictionary containing `Example`'s constructor parameters, AND `BaseClass`'s (it's parent class) constructor parameters, organised by class:

```
>>> get_function_params(Example, True)
{
    <class '__main__.Example'>: {
        'd': <Parameter "d">, 'e': <Parameter "e='hello'">, 'f': <Parameter
↪"f=None">,
        'a': <Parameter "a='overridden'">
    },
    <class '__main__.BaseClass'>: {'a': <Parameter "a">, 'b': <Parameter "b">, 'c
↪': <Parameter "c=1234">}
}
```

We can also add the optional kwarg `merge=True`, which merges the parameters of the originally passed class, and it's parents.

This is done in reverse order, so that children's conflicting constructor parameters take priority over their parents, as can be seen below with `a` which is shown as `a='overridden'` - the overridden parameter of the class `Example` with a default value, instead of the parent's `a` which makes `a` mandatory:

```
>>> get_function_params(Example, True, merge=True)
{
    'a': <Parameter "a='overridden'">, 'b': <Parameter "b">, 'c': <Parameter
↪"c=1234">,
    'd': <Parameter "d">, 'e': <Parameter "e='hello'">, 'f': <Parameter "f=None">
}
```

Parameters

- **obj** (*type|callable*) – A class (not an instance) or callable (function / lambda) to extract and filter the parameter's from. If a class is passed, the parameters of the constructor will be returned (`__init__`), excluding the initial `self` parameter.
- **check_parents** (*bool*) – (Default: `False`) If `obj` is a class and this is True, will recursively grab the constructor parameters for all parent classes, and return the parameters as a dictionary of `{<class X>: {'a': <Parameter 'a'>}, <class Y>: {'b': <Parameter 'b'>}`, unless `merge` is also set to `True`.

Key bool ignore_xargs (Default: `True`) Filter out any catch-all positional arguments (e.g. `*args`)

Key bool ignore_xkwargs (Default: `True`) Filter out any catch-all keyword arguments (e.g. `**kwargs`)

Key bool ignore_defaults (Default: `False`) Filter out any parameter which has a default value (e.g. args usable as kwargs)

Key bool ignore_positional (Default: `False`) Filter out any parameter which doesn't have a default value (mandatory args)

Key bool merge (Default: `False`) If this is True, when `check_parents` is enabled, all parameters will be flatted into a singular dictionary, e.g. `{'a': <Parameter 'a'>, 'b': <Parameter "b">}`

Returns

3.6.19 human_name

`privex.helpers.common.`**`human_name`**(*class_name: Union[str, bytes, callable, Type[object]]*) → str

This function converts a class/function name into a Title Case name. It also directly accepts classes/functions.

Input names can be either snake case `my_function`, or InitialCaps `MyClass` - though mixtures of the two may work, such as `some_functionName` - however `some_FunctionName` will not (causes double spaces).

Examples

Using a plain string or bytes:

```
>>> human_name(b'_some_functionName')
'Some Function Name'
>>> human_name('SomeClassName')
'Some Class Name'
```

Using a reference to a function:

```
>>> def some_func():
...     pass
>>> human_name(some_func)
'Some Func'
```

Using a reference to a class, or an instance of a class:

```
>>> class MyExampleClass:
...     pass
>>> my_instance = MyExampleClass()
>>> human_name(MyExampleClass)
'My Example Class'
>>> human_name(my_instance)
'My Example Class'
```

Parameters **class_name** – The name of a class/function specified either in InitialCaps or snake_case. You may also pass a function reference, class reference, or class instance. (see examples)

Return str human_name The humanised Title Case name of `class_name`

3.6.20 inject_items

`privex.helpers.common.`**`inject_items`**(*items: list*, *dest_list: list*, *position: int*) → List[str]

Inject a list `items` after a certain element in `dest_list`.

NOTE: This does NOT alter `dest_list` - it returns a **NEW list** with `items` injected after the given `position` in `dest_list`.

Example Usage:

```
>>> x = ['a', 'b', 'e', 'f', 'g']
>>> y = ['c', 'd']
>>> # Inject the list 'y' into list 'x' after element 1 (b)
>>> inject_items(y, x, 1)
['a', 'b', 'c', 'd', 'e', 'f', 'g']
```

Parameters

- **items** (*list*) – A list of items to inject into `dest_list`
- **dest_list** (*list*) – The list to inject `items` into
- **position** (*int*) – Inject `items` after this element (0 = 1st item) in `dest_list`

Return List[str] injected `dest_list` with the passed `items` list injected at `position`

3.6.21 io_tail

privex.helpers.common.**io_tail**(*f: BinaryIO*, *nlines: int = 20*, *bsz: int = 4096*) → Generator[List[str], None, None]

NOTE: If you're only loading a small amount of lines, e.g. less than 1MB, consider using the much easier *tail()* function - it only requires one call and returns the lines as a singular, correctly ordered list.

This is a generator function which works similarly to `tail` on UNIX systems. It efficiently retrieves lines in reverse order using the passed file handle `f`.

WARNING: This function is a generator which returns "chunks" of lines - while the lines within each chunk are in the correct order, the chunks themselves are backwards, i.e. each chunk retrieves lines prior to the previous chunk.

This function was designed as a generator to allow for **memory efficient handling of large files**, and tailing large amounts of lines. It only loads `bsz` bytes from the file handle into memory with each iteration, allowing you to process each chunk of lines as they're read from the file, instead of having to load all `nlines` lines into memory at once.

To ensure your retrieved lines are in the correct order, with each iteration you must PREPEND the outputted chunk to your final result, rather than APPEND. Example:

```
>>> from privex.helpers import io_tail
>>> lines = []
>>> with open('/tmp/example', 'rb') as fp:
...     # We prepend each chunk from 'io_tail' to our result variable 'lines'
...     for chunk in io_tail(fp, nlines=10):
...         lines = chunk + lines
>>> print('\n'.join(lines))
```

Modified to be more memory efficient, but originally based on this SO code snippet: https://stackoverflow.com/a/136354

Parameters

- **f** (*BinaryIO*) – An open file handle for the file to tail, must be in **binary mode** (e.g. `rb`)
- **nlines** (*int*) – Total number of lines to retrieve from the end of the file
- **bsz** (*int*) – Block size (in bytes) to load with each iteration (default: 4096 bytes). DON'T CHANGE UNLESS YOU UNDERSTAND WHAT THIS MEANS.

Return Generator chunks Generates chunks (in reverse order) of correctly ordered lines as `List[str]`

3.6.22 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`.

3.6.23 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`.

3.6.24 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

- **line** (*str*) – A string of columns separated by commas e.g. `hello,world,foo`
- **csvsplit** (*str*) – A character (or several) used to terminate each value in the list. Default: comma `,`

3.6.25 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()

3.6.26 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)

3.6.27 reverse_io

`privex.helpers.common.`**`reverse_io`**(*f: BinaryIO*, *blocksize: int = 4096*) → Generator[bytes, None, None]

Read file as series of blocks from end of file to start.

The data itself is in normal order, only the order of the blocks is reversed. ie. "hello world" -> ["ld","wor", "lo ", "hel"] Note that the file must be opened in binary mode.

Original source: https://stackoverflow.com/a/136354

3.6.28 shell_quote

`privex.helpers.common.`**`shell_quote`**(**args: str*) → str

Takes command line arguments as positional args, and properly quotes each argument to make it safe to pass on the command line. Outputs a string containing all passed arguments properly quoted.

Uses `shlex.join()` on Python 3.8+, and a for loop of `shlex.quote()` on older versions.

Example:

```
>>> print(shell_quote('echo', '"orange"'))
echo '"orange"'
```

3.6.29 stringify

`privex.helpers.common.`**`stringify`**(*data: Optional[Union[str, bytes]]*, *encoding='utf-8'*, *if_none=None*) → str

Convert a piece of data into a string (from bytes) if it isn't already:

```
>>> stringify(b"hello world")
"hello world"
```

By default, if `data` is `None`, then `None` will be returned.

If you'd rather convert `None` into a blank string, use `if_node=""`, like so:

```
>>> repr(stringify(None))
'None'
>>> stringify(None, if_none="")
''
```

3.6.30 tail

privex.helpers.common.**tail**(*filename: str*, *nlines: int = 20*, *bsz: int = 4096*) → List[str]

Pure python equivalent of the UNIX `tail` command. Simply pass a filename and the number of lines you want to load from the end of the file, and a `List[str]` of lines (in forward order) will be returned.

This function is simply a wrapper for the highly efficient *io_tail()*, designed for usage with a small (<10,000) amount of lines to be tailed. To allow for the lines to be returned in the correct order, it must load all `nlines` lines into memory before it can return the data.

If you need to `tail` a large amount of data, e.g. 10,000+ lines of a logfile, you should consider using the lower level function *io_tail()* - which acts as a generator, only loading a certain amount of bytes into memory per iteration.

Example file `/tmp/testing`:

```
this is an example 1
this is an example 2
this is an example 3
this is an example 4
this is an example 5
this is an example 6
```

Example usage:

```
>>> from privex.helpers import tail
>>> lines = tail('/tmp/testing', nlines=3)
>>> print("\n".join(lines))
this is an example 4
this is an example 5
this is an example 6
```

Parameters

- **filename** (*str*) – Path to file to tail. Relative or absolute path. Absolute path is recommended for safety.
- **nlines** (*int*) – Total number of lines to retrieve from the end of the file
- **bsz** (*int*) – Block size (in bytes) to load with each iteration (default: 4096 bytes). DON'T CHANGE UNLESS YOU UNDERSTAND WHAT THIS MEANS.

Return List[str] lines The last 'nlines' lines of the file 'filename' - in forward order.

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.

3.6.31 ErrHelpParser

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='+')
```

__init__(*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*)

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

3.6.31.1 Methods

Methods

error(message)	Prints a usage message incorporating the message to stderr and exits.

3.6.31.1.1 error

ErrHelpParser.**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.

3.7 privex.helpers.collections

Functions, classes and/or types which either **are**, or are related to Python variable storage types (`dict`, `tuple`, `list`, `set` etc.)

3.7.1 Object-like Dictionaries (dict's)

Have you ever wanted a dictionary that works like an object, where you can get/set dictionary keys using attributes (`x.something`) as easily as you can with items (`x['something']`)?

We did. So we invented *DictObject*, a sub-class of the built-in `dict`, making it compatible with most functions/methods which expect a `dict` (e.g. `json.dumps()`).

You can create a new *DictObject* and use it just like a `dict`, or you can convert an existing `dict` into a `DictObject` much like you'd cast any other builtin type.

It can also easily be cast back into a standard `dict` when needed, without losing any data.

3.7.1.1 Creating a new DictObject and using it

Since *DictObject* is a subclass of the builtin `dict`, you can instantiate a new *DictObject* in the same way you would use the standard `dict` class:

```
>>> d = DictObject(hello='world')
>>> d
{'hello': 'world'}
>>> d['hello']
'world'
>>> d.hello
'world'
>>> d.lorem = 'ipsum'
>>> d['orange'] = 'banana'
>>> d
{'hello': 'world', 'lorem': 'ipsum', 'orange': 'banana'}
```

3.7.1.2 Converting an existing dictionary (dict) into a DictObject

You can convert an existing `dict` into a *DictObject* in the same way you'd convert any other object into a `dict`:

```
>>> y = {"hello": "world", "example": 123}
>>> x = DictObject(y)
>>> x.example
123
>>> x['hello']
'world'
>>> x.hello = 'replaced'
>>> x
{'hello': 'replaced', 'example': 123}
```

It also works vice versa, you can convert a *DictObject* instance back into a `dict` just as easily as you converted the *dict* into a *DictObject*.

```
>>> z = dict(x)
>>> z
{'hello': 'replaced', 'example': 123}
```

3.7.2 Dict-able NamedTuple's

While `collections.namedtuple()`'s can be useful, they have some quirks, such as not being able to access fields by item/key (`x['something']`). They also expose a method `._asdict()`, but cannot be directly casted into a `dict` using `dict(x)`.

Our *dictable_namedtuple()* collection is designed to fix these quirks.

3.7.2.1 What is a dictable_namedtuple and why use it?

Unlike the normal `namedtuple()` types, ``dictable_namedtuple``s add extra convenience functionality:

- Can access fields via item/key: `john['first_name']`
- Can convert instance into a dict simply by casting: `dict(john)`
- Can set new items/attributes on an instance, even if they weren't previously defined.
- NOTE: You cannot edit an original namedtuple field defined on the type, those remain read only

There are three functions available for working with `dictable_namedtuple` classes/instances, each for different purposes.

- *`dictable_namedtuple()`* - Create a new `dictable_namedtuple` type for instantiation.
- *`convert_dictable_namedtuple()`* - Convert an existing **namedtuple instance** (not a type/class) into a `dictable_namedtuple` instance.
- *`subclass_dictable_namedtuple()`* - Convert an existing **namedtuple type/class** (not an instance) into a `dictable_namedtuple` type for instantiation.

3.7.2.2 Importing dictable_namedtuple functions

```
from collections import namedtuple
from privex.helpers import dictable_namedtuple, convert_dictable_namedtuple, subclass_
↪dictable_namedtuple
```

3.7.2.3 Creating a NEW dictable_namedtuple type and instance

If you're **creating a new Named Tuple**, and you want it to support dictionary-like access, and have it able to be converted into a dict simply through `dict(my_namedtuple)`, then you want *`dictable_namedtuple()`*

```
Person = dictable_namedtuple('Person', 'first_name last_name')
john = Person('John', 'Doe')
dave = Person(first_name='Dave', last_name='Smith')
print(dave['first_name'])       # Prints:  Dave
print(dave.first_name)          # Prints:  Dave
print(john[1])                  # Prints:  Doe
print(dict(john))               # Prints:  {'first_name': 'John', 'last_name': 'Doe'}
```

3.7.2.4 Converting an existing namedtuple instance into a dictable_namedtuple instance

If you have **existing Named Tuple instances**, e.g. returned from a python library, then you can use *`convert_dictable_namedtuple()`* to convert them into `dictable_namedtuple`'s and gain all the functionality mentioned at the start of this section.

```
Person = namedtuple('Person', 'first_name last_name')  # This is an existing␣
↪namedtuple "type" or "class"
john = Person('John', 'Doe')  # This is an existing namedtuple instance
john.first_name               # This works on a standard namedtuple. Returns: John
john[1]                       # This works on a standard namedtuple. Returns: Doe
john['first_name']            # However, this would throw a TypeError.
dict(john)                    # And this would throw a ValueError.
```

(continues on next page)

(continued from previous page)

```
# We can now convert 'john' into a dictable_namedtuple, which will retain the␣
↪functionality of a
# namedtuple, but add to the functionality by allowing dict-like key access, updating/
↪creating new
# fields, as well as painlessly casting to a dictionary.

d_john = convert_dictable_namedtuple(john)
d_john.first_name               # Returns: John
d_john[1]                       # Returns: Doe
d_john['first_name']            # Returns: 'John'
dict(d_john)                    # Returns: {'first_name': 'John', 'last_name': 'Doe'}
```

3.7.2.5 Converting an existing namedtuple type/class into a dictable_namedtuple type/class

If you have **existing Named Tuple type/class** then you can use *subclass_dictable_namedtuple()* to convert the type/class into a `dictable_namedtuple` type/class and gain all the functionality mentioned at the start of this section. (**NOTE:** it's usually easier to just replace your `namedtuple` calls with `dictable_namedtuple`)

```
Person = namedtuple('Person', 'first_name last_name')  # This is an existing␣
↪namedtuple "type" or "class"
# We can now convert the 'Person' type into a dictable_namedtuple type.
d_Person = subclass_dictable_namedtuple(Person)
# Then we can use this converted type to create instances of Person with dictable_
↪namedtuple functionality.
john = d_Person('John', 'Doe')
john.first_name               # Returns: John
john[1]                       # Returns: Doe
john['first_name']            # Returns: 'John'
dict(john)                    # Returns: {'first_name': 'John', 'last_name': 'Doe'}
```

Copyright:

```
    +==================================================+
    |                © 2019 Privex Inc.                |
    |              https://www.privex.io               |
    +==================================================+
    |                                                  |
    |        Originally Developed by Privex Inc.       |
    |        License: X11 / MIT                        |
    |                                                  |
    |        Core Developer(s):                        |
    |                                                  |
    |          (+)  Chris (@someguy123) [Privex]       |
    |          (+)  Kale (@kryogenic) [Privex]         |
    |                                                  |
    +==================================================+

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

Functions

convert_dictable_namedtuple(nt_instance[, ...])	Convert an existing collections.namedtuple() instance into a dictable_namedtuple instance.
dictable_namedtuple(typename, field_names, ...)	Creates a dictable_namedtuple type for instantiation (same usage as collections.namedtuple()) - unlike namedtuple, dictable_namedtuple instances allow item (dict-like) field access, support writing and can be painlessly converted into dictionaries via dict(my_namedtuple).
is_namedtuple(*objs)	Takes one or more objects as positional arguments, and returns True if ALL passed objects are namedtuple instances
make_dict_tuple(typename, field_names, ...)	Generates a collections.namedtuple() type, with added / modified methods injected to make it into a dictable_namedtuple.
subclass_dictable_namedtuple(named_type[, ...])	Convert an existing collections.namedtuple() **type** into a dictable_namedtuple.

3.7.2.5.1 convert_dictable_namedtuple

privex.helpers.collections.**convert_dictable_namedtuple**(*nt_instance*, *typename=None*, *module=None*, ***kwargs*) → Union[NamedTuple, Dict]

Convert an existing collections.namedtuple() instance into a dictable_namedtuple instance.

Example

First we create a namedtuple type Person

```
>>> from collections import namedtuple
>>> Person = namedtuple('Person', 'first_name last_name')
```

Next we create an instance of Person called John Doe, and we can confirm it's a normal namedtuple, as we can't access first_name by item/key.

```
>>> john = Person('John', 'Doe')
>>> john['first_name']
TypeError: tuple indices must be integers or slices, not str
```

Using *convert_dictable_namedtuple()*, we can convert john from a normal namedtuple, into a dictable_namedtuple.

This enables many convenience features (see *dictable_namedtuple()* for more info) such as easy casting to a dict, and accessing fields by item/key (square brackets):

```
>>> from privex.helpers import convert_dictable_namedtuple
>>> d_john = convert_dictable_namedtuple(john)
>>> d_john
Person(first_name='John', last_name='Doe')
>>> d_john['first_name']
```

(continues on next page)

(continued from previous page)

```
'John'
>>> dict(d_john)
{'first_name': 'John', 'last_name': 'Doe'}
```

Parameters

- **nt_instance** – An instantiated namedtuple object (using a type returned from `collections.namedtuple()`)
- **typename** (*str*) – Optionally, you can change the name of your instance's class, e.g. if you provide a `Person` instance, but you set this to `Man`, then this will return a `Man` instance, like so: `Man(first_name='John', last_name='Doe')`
- **module** (*str*) – Optionally, you can change the module that the type class belongs to. Otherwise it will inherit the module path from the class of your instance.

Key bool read_only (Default: `False`) If set to `True`, the outputted dictable_namedtuple instance will not allow new fields to be created via attribute / item setting.

Return dictable_namedtuple The instance you passed `nt_instance`, converted into a dictable_namedtuple

3.7.2.5.2 dictable_namedtuple

privex.helpers.collections.**dictable_namedtuple**(*typename*, *field_names*, **args*, ***kwargs*) → Union[Type[collections.namedtuple], dict]

Creates a dictable_namedtuple type for instantiation (same usage as `collections.namedtuple()`) - unlike namedtuple, dictable_namedtuple instances allow item (dict-like) field access, support writing and can be painlessly converted into dictionaries via `dict(my_namedtuple)`.

Named tuple instances created from `dictable_namedtuple` types are generally backwards compatible with any code that expects a standard `collections.namedtuple()` type instance.

Quickstart

```
>>> from privex.helpers import dictable_namedtuple
>>> # Define a dictable_namedtuple type of 'Person', which has two fields - first_
↪name and last_name
>>> p = dictable_namedtuple('Person', 'first_name last_name')
>>> john = p('John', 'Doe')     # Alternatively you can do p(first_name='John',␣
↪last_name='Doe')
>>> john.first_name             # You can retrieve keys either via attributes (dot␣
↪notation)
'John'
>>> john['last_name']           # Via named keys (square brackets)
'Doe'
>>> john[1]                     # Or, via indexed keys (square brackets, with␣
↪integer keys)
'Doe'
>>> john.middle_name = 'Davis' # You can also update / set new keys via attribute/
↪key/index
>>> dict(john)                  # Newly created keys will show up as normal in␣
↪dict(your_object)
{'first_name': 'John', 'last_name': 'Doe', 'middle_name': 'Davis'}
```

(continues on next page)

(continued from previous page)

```
>>> john                        # As well as in the representation in the REPL or␣
↪when str() is called.
Person(first_name='John', last_name='Doe', middle_name='Davis')
```

This function adds / overrides the following methods on the generated namedtuple type:

- _asdict
- __iter__
- __getitem__
- __getattribute__
- __setitem__
- __setattr__
- __repr__

Extra functionality compared to the standard `namedtuple()` generated classes:

- Can access fields via item/key: `john['first_name']`
- Can convert instance into a dict simply by casting: `dict(john)`
- Can set new items/attributes on an instance, even if they weren't previously defined. `john['middle_name'] = 'Davis'` or `john.middle_name = 'Davis'`

Example Usage

First we'll create a named tuple typle called `Person`, which takes two arguments, first_name and last_name.

```
>>> from privex.helpers import dictable_namedtuple
>>> Person = dictable_namedtuple('Person', 'first_name last_name')
```

Now we'll create an instance of `Person` called `john`. These instances look like normal `namedtuple`'s, and should be generally compatible with any functions/methods which deal with named tuple's.

```
>>> john = Person('John', 'Doe')    # Alternatively you can do Person(first_name=
↪'John', last_name='Doe')
>>> john
Person(first_name='John', last_name='Doe')
```

Unlike a normal `namedtuple` type instance, we can access fields by attribute (`.first_name`), index (`[0]`), AND by item/key name (`['last_name']`).

```
>>> john.first_name
'John'
>>> john[0]
'John'
>>> john['last_name']
'Doe'
```

Another potentially useful feature, is that you can also update / create new fields, via your preferred method of field notation (other than numbered indexes, since those don't include a field name):

```
>>> john['middle_name'] = 'Davis'
>>> john.middle_name = 'Davis'
```

We can also convert `john` into a standard dictionary, with a simple `dict(john)` cast. You can see that the new field we added (`middle_name`) is present in the dictionary serialized format.

```
>>> dict(john)
{'first_name': 'John', 'last_name': 'Doe', 'middle_name': 'Davis'}
```

Parameters

- **typename** (*str*) – The name used for the namedtuple type/class
- **field_names** (*str*) – One or more field names separated by spaces, e.g. `'id first_name last_name address'`

Key bool read_only (Default: `False`) If set to `True`, the outputted dictable_namedtuple instance will not allow new fields to be created via attribute / item setting.

Return Type[namedtuple] dict_namedtuple A dict_namedtuple type/class which can be instantiated with the given `field_names` via positional or keyword args.

3.7.2.5.3 is_namedtuple

`privex.helpers.collections.`**`is_namedtuple`**(**objs*) → bool

Takes one or more objects as positional arguments, and returns `True` if ALL passed objects are namedtuple instances

Example usage

First, create or obtain one or more NamedTuple objects:

```
>>> from collections import namedtuple

>>> Point, Person = namedtuple('Point', 'x y'), namedtuple('Person', 'first_name␣
↪last_name')

>>> pt1, pt2 = Point(1.0, 5.0), Point(2.5, 1.5)
>>> john = Person('John', 'Doe')
```

We'll also create a `tuple`, `dict`, and `str` to show they're detected as invalid:

```
>>> normal_tuple, tst_dict, tst_str = (1, 2, 3,), dict(hello='world'), "hello␣
↪world"
```

First we'll call *`is_namedtuple()`* with our Person NamedTuple object `john`:

```
>>> is_namedtuple(john)
True
```

As expected, the function shows `john` is in-fact a named tuple.

Now let's try it with our two Point named tuple's `pt1` and `pt2`, plus our Person named tuple `john`.

```
>>> is_namedtuple(pt1, john, pt2)
True
```

Since all three arguments were named tuples (even though pt1/pt2 and john are different types), the function returns `True`.

Now we'll test with a few objects that clearly aren't named tuple's:

```
>>> is_namedtuple(tst_str)    # Strings aren't named tuples.
False
>>> is_namedtuple(normal_tuple)     # A plain bracket tuple is not a named tuple.
False
>>> is_namedtuple(john, tst_dict)  # ``john`` is a named tuple, but a dict isn't,␣
↪thus False is returned.
False
```

Original source: https://stackoverflow.com/a/2166841

Parameters **objs** (*Any*) – The objects (as positional args) to check whether they are a NamedTuple

Return bool is_namedtuple `True` if all passed `objs` are named tuples.

3.7.2.5.4 make_dict_tuple

privex.helpers.collections.**make_dict_tuple**(*typename*, *field_names*, **args*, ***kwargs*)

Generates a `collections.namedtuple()` type, with added / modified methods injected to make it into a `dictable_namedtuple`.

Note: You probably want to be using *`dictable_namedtuple()`* instead of calling this directly.

3.7.2.5.5 subclass_dictable_namedtuple

privex.helpers.collections.**subclass_dictable_namedtuple**(*named_type: type*, *typename=None*, *module=None*, ***kwargs*) → type

Convert an existing `collections.namedtuple()` **type** into a dictable_namedtuple.

If you have an INSTANCE of a type (e.g. it has data attached), use *`convert_dictable_namedtuple()`*

Example:

```
>>> from collections import namedtuple
>>> from privex.helpers import subclass_dictable_namedtuple
>>> # Create a namedtuple type called 'Person'
>>> orig_Person = namedtuple('Person', 'first_name last_name')
>>> # Convert the 'Person' type into a dictable_namedtuple
>>> Person = subclass_dictable_namedtuple(orig_Person)
>>> john = Person('John', 'Doe')    # Create an instance of this dictable_
↪namedtuple Person
>>> john['middle_name'] = 'Davis'
```

Parameters

- **named_type** (*type*) – A NamedTuple type returned from `collections.namedtuple()`
- **typename** (*str*) – Optionally, you can change the name of your type, e.g. if you provide a `Person` class type, but you set this to `Man`, then this will return a `Man` class type.
- **module** (*str*) – Optionally, you can change the module that the type class belongs to. Otherwise it will inherit the module path from `named_type`.

Key bool read_only (Default: `False`) If set to `True`, the outputted dictable_namedtuple type will not allow new fields to be created via attribute / item setting.

Return type dictable_namedtuple Your `named_type` converted into a dictable_namedtuple type class.

Classes

DictObject	A very simple `dict` wrapper, which allows you to read and write dictionary keys using attributes (dot notation) PLUS standard item (key / square bracket notation) access.
Dictable()	A small abstract class for use with Python 3.7 dataclasses.
Mocker(modules, attributes)	This mock class is designed to be used either to act as a stand-in "noop" (no operation) object, which could be used either as a drop-in replacement for a failed module / class import, or for certain unit tests.
OrderedDictObject	Ordered version of *DictObject* - dictionary with attribute access.

3.7.2.5.6 DictObject

class privex.helpers.collections.**DictObject**

A very simple `dict` wrapper, which allows you to read and write dictionary keys using attributes (dot notation) PLUS standard item (key / square bracket notation) access.

Example Usage (creating and using a new DictObject):

```
>>> d = DictObject(hello='world')
>>> d
{'hello': 'world'}
>>> d['hello']
'world'
>>> d.hello
'world'
>>> d.lorem = 'ipsum'
>>> d['orange'] = 'banana'
>>> d
{'hello': 'world', 'lorem': 'ipsum', 'orange': 'banana'}
```

Example Usage (converting an existing dict):

```
>>> y = {"hello": "world", "example": 123}
>>> x = DictObject(y)
>>> x.example
123
>>> x['hello']
'world'
>>> x.hello = 'replaced'
>>> x
{'hello': 'replaced', 'example': 123}
```

__init__(**args*, ***kwargs*)

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

3.7.2.5.6.1 Methods

Methods

3.7.2.5.7 Dictable

class privex.helpers.collections.**Dictable**

A small abstract class for use with Python 3.7 dataclasses.

Allows dataclasses to be converted into a `dict` using the standard `dict()` function:

```
>>> @dataclass
>>> class SomeData(Dictable):
...     a: str
...     b: int
...
>>> mydata = SomeData(a='test', b=2)
>>> dict(mydata)
{'a': 'test', 'b': 2}
```

Also allows creating dataclasses from arbitrary dictionaries, while ignoring any extraneous dict keys.

If you create a dataclass using a `dict` and you have keys in your `dict` that don't exist in the dataclass, it'll generally throw an error due to non-existent kwargs:

```
>>> mydict = dict(a='test', b=2, c='hello')
>>> sd = SomeData(**mydict)
TypeError: __init__() got an unexpected keyword argument 'c'
```

Using `from_dict` you can simply trim off any extraneous dict keys:

```
>>> sd = SomeData.from_dict(**mydict)
>>> sd.a, sd.b
('test', 2)
>>> sd.c
AttributeError: 'SomeData' object has no attribute 'c'
```

__init__()

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

3.7.2.5.7.1 Methods

Methods

---

from_dict(env)

3.7.2.5.7.2 from_dict

classmethod Dictable.**from_dict**(*env*)

3.7.2.5.8 Mocker

class privex.helpers.collections.**Mocker**(*modules: dict = None*, *attributes: dict = None*)

This mock class is designed to be used either to act as a stand-in "noop" (no operation) object, which could be used either as a drop-in replacement for a failed module / class import, or for certain unit tests.

If you need additional functionality such as methods having actual behaviour, you can set attributes on a Mocker instance to either a lambda, or point them at a real function/method:

```
>>> m = Mocker()
>>> m.some_func = lambda a: a+1
>>> m.some_func(5)
6
```

Example use case - fallback for unimportant module imports

Below is a real world example of using *Mocker* and *privex.helpers.decorators.mock_decorator()* to simulate `pytest` - allowing your tests to run under the standard `unittest` framework if a user doesn't have pytest (as long as your tests aren't critically dependent on PyTest).

Try importing `pytest` then fallback to a mock pytest:

```
>>> try:
...     import pytest
... except ImportError:
...     from privex.helpers import Mocker, mock_decorator
...     print('Failed to import pytest. Using privex.helpers.Mocker to fake␣
↪pytest.')
...     # Make pytest pretend to be the class 'module' (the class actually used␣
↪for modules)
...     pytest = Mocker.make_mock_class('module')
...     # To make pytest.mark.skip work, we add the fake module 'mark', then set␣
↪skip to `mock_decorator`
...     pytest.add_mock_module('mark')
...     pytest.mark.skip = mock_decorator
...
```

Since we added the mock module `mark`, and set the attribute `skip` to point at `mock_decorator`, the test function `test_something` won't cause a syntax error. `mock_decorator` will just call test_something() which doesn't do anything anyway:

```
>>> @pytest.mark.skip(reason="this test doesn't actually do anything...")
... def test_something():
...     pass
>>>
>>> def test_other_thing():
...     if True:
...         return pytest.skip('cannot test test_other_thing because of an error')
...
>>>
```

Generating "disguised" mock classes

If you need the mock class to appear to have a certain class name and/or module path, you can generate "disguised" mock classes using *make_mock_class()* like so:

```
>>> redis = Mocker.make_mock_class('Redis', module='redis')
>>> redis
<redis.Redis object at 0x7fd7402ea4a8>
```

A :class:`.Mocker` instance has the following behaviour

- Attributes that don't exist result in a function being returned, which accepts any arguments / keyword args, and simply returns `None`

Example:

```
>>> m = Mocker()
>>> repr(m.randomattr('hello', world=123))
'None'
```

- Arbitrary attributes `x.something` and items `x['something']` can be set on an instance, and they will be similarly returned when they're accessed. Attributes and items share the same key/value's, so the following examples are all accessing the same data:

Example:

```
>>> m = Mocker()
>>> m.example = 'hello'
>>> m['example'] = 'world'
>>> print(m.example)
world
>>> print(m['example'])
world
```

- You can add arbitrary "modules" to a Mocker instance. With only the `name` argument, *add_mock_module()* will add a "module" under the instance, which is really just another *Mocker* instance.

Example:

```
>>> m = Mocker()
>>> m.add_mock_module('my_module')
>>> m.my_module.example = 'hello'
>>> print(m.my_module['example'], m.my_module.example)
hello hello
```

__init__(*modules: dict = None*, *attributes: dict = None*)
Initialize self. See help(type(self)) for accurate signature.

3.7.2.5.8.1 Methods

Methods

__init__([modules, attributes])	Initialize self.
add_mock_module(name[, value, mock_attrs, ...])	Add a fake sub-module to this Mocker instance.
make_mock_class([name, instance])	Return a customized mock class or create an instance which appears to be named `name`

3.7.2.5.8.2 __init__

`Mocker.`**`__init__`**(*modules: dict = None*, *attributes: dict = None*)

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

3.7.2.5.8.3 add_mock_module

`Mocker.`**`add_mock_module`**(*name: str*, *value=None*, *mock_attrs: dict = None*, *mock_modules: dict = None*)

Add a fake sub-module to this Mocker instance.

Example:

```
>>> m = Mocker()
>>> m.add_mock_module('my_module')
>>> m.my_module.example = 'hello'
>>> print(m.my_module['example'], m.my_module.example)
hello hello
```

Parameters

- **`name`** (*str*) – The name of the module to add.
- **`value`** – Set the "module" to this object, instead of an instance of *Mocker*
- **`mock_attrs`** (*dict*) – If `value` is `None`, then this can optionally contain a dictionary of attributes/items to pre-set on the Mocker instance.
- **`mock_modules`** (*dict*) – If `value` is `None`, then this can optionally contain a dictionary of "modules" to pre-set on the Mocker instance.

3.7.2.5.8.4 make_mock_class

`classmethod` `Mocker.`**`make_mock_class`**(*name='Mocker'*, *instance=True*, ***kwargs*)

Return a customized mock class or create an instance which appears to be named `name`

Allows code which might check `x.__class__.__name__` to believe it's the correct object.

Using the kwarg `module` you can change the module that the class / instance appears to have been imported from, allowing for quite deceiving fake classes and instances.

Example usage:

```
>>> redis = Mocker.make_mock_class('Redis', module='redis')
>>> # As seen below, the class appears to be called Redis, and even claims to be␣
↪from the module `redis`
>>> redis
<redis.Redis object at 0x7fd7402ea4a8>
>>> print(f'Module: {redis.__module__} - Class Name: {redis.__class__.__name__}')
Module: redis - Class Name: Redis
```

Creating methods/attributes dynamically

You can set arbitrary attributes to point at a function, or just set them to a lambda:

```
>>> redis.exists = lambda key: 1
>>> redis.exists('hello')
1
>>> redis.hello()  # Non-existent attributes just act as a function that eats any␣
↪args and returns None
None
```

Parameters

- **name** – The name to write onto the mock class's `__name__` (and `__qualname__` if not specified)
- **instance** (*bool*) – If `True` then the disguised mock class will be returned as an instance. Otherwise the raw class itself will be returned for you to instantiate yourself.
- **kwargs** – All kwargs (other than `qualname`) are forwarded to `__init__` of the disguised class if `instance` is True.

Key str qualname Optionally specify the "qualified name" to insert into `__qualname__`. If this isn't specified, then `name` is used for qualname, which is fine for most cases anyway.

Key str module Optionally override the module namespace that the class is supposedly from. If not specified, then the class will just inherit this module (`privex.helpers.common`)

Returns

3.7.2.5.9 OrderedDictObject

class privex.helpers.collections.**OrderedDictObject**

Ordered version of *DictObject* - dictionary with attribute access. See *DictObject*

__init__(**args*, ***kwargs*)

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

3.7.2.5.9.1 Methods

Methods

—

3.8 privex.helpers.converters

Various functions/classes which convert/parse objects from one type into another.

Copyright:

```
    +==================================================+
    |                 © 2019 Privex Inc.               |
    |               https://www.privex.io              |
    +==================================================+
    |                                                  |
    |        Originally Developed by Privex Inc.       |
    |        License: X11 / MIT                        |
    |                                                  |
    |        Core Developer(s):                        |
    |                                                  |
    |          (+)  Chris (@someguy123) [Privex]       |
    |          (+)  Kale (@kryogenic) [Privex]         |
    |                                                  |
    +==================================================+

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

Functions

convert_bool_int(d[, if_empty, fail_empty])	Convert a boolean `d` into an integer (`0` for `False`, `1` for `True`)
convert_datetime(d[, if_empty, fail_empty])	Convert the object `d` into a `datetime.datetime` object.
convert_int_bool(d[, if_empty, fail_empty])	Convert an integer `d` into a boolean (`0` for `False`, `1` for `True`)
convert_unixtime_datetime(d[, if_empty, ...])	Convert a unix timestamp into a `datetime.datetime` object

3.8.1 convert_bool_int

privex.helpers.converters.**convert_bool_int**(*d*, *if_empty=0*, *fail_empty=False*) → int

Convert a boolean `d` into an integer (`0` for `False`, `1` for `True`)

3.8.2 convert_datetime

privex.helpers.converters.**convert_datetime**(*d*, *if_empty=None*, *fail_empty=False*, ***kwargs*) → Optional[datetime.datetime]

Convert the object `d` into a `datetime.datetime` object.

If `d` is a string or bytes, then it will be parsed using `dateutil.parser.parse()`

If `d` is an int/float/Decimal, then it will be assumed to be a unix epoch timestamp.

Examples:

```
>>> convert_datetime("2019-01-01T00:00:00Z")          # ISO date/time
datetime.datetime(2019, 1, 1, 0, 0, tzinfo=tzutc())

>>> convert_datetime("01/JAN/2019 00:00:00.0000")     # Human date/time with␣
↪month name
datetime.datetime(2019, 1, 1, 0, 0, tzinfo=tzutc())

>>> convert_datetime(1546300800)                      # Unix timestamp as integer
datetime.datetime(2019, 1, 1, 0, 0, tzinfo=tzutc())

>>> convert_datetime(1546300800000)                   # Unix timestamp␣
↪(milliseconds) as integer
datetime.datetime(2019, 1, 1, 0, 0, tzinfo=tzutc())
```

Parameters

- **d** – Object to convert into a datetime
- **if_empty** – If `d` is empty / None, return this value
- **fail_empty** (*bool*) – (Def: `False`) If this is True, then if `d` is empty, raises `AttributeError`

Key datetime.tzinfo tzinfo (Default: `dateutil.tz.tzutc`) If no timezone was detected by the parser, use this timezone. Set this to `None` to disable forcing timezone-aware dates.

Raises

- **AttributeError** – When `d` is empty and `fail_empty` is set to True.
- **dateutil.parser.ParserError** – When `d` could not be parsed into a date.

Return datetime converted The converted `datetime.datetime` object.

3.8.3 convert_int_bool

privex.helpers.converters.**convert_int_bool**(*d*, *if_empty=False*, *fail_empty=False*) → bool

Convert an integer `d` into a boolean (`0` for `False`, `1` for `True`)

3.8.4 convert_unixtime_datetime

privex.helpers.converters.**convert_unixtime_datetime**(*d: Union[str, int, float, decimal.Decimal]*, *if_empty=None*, *fail_empty=False*) → datetime.datetime

Convert a unix timestamp into a `datetime.datetime` object

3.9 privex.helpers.crypto

Cryptography related helper classes/functions

Dependencies

Requires the `cryptography` Python package:

```
# Either install privex-helpers with the `crypto` extra
pipenv install 'privex-helpers[crypto]'  # Using pipenv if you have it
pip3 install 'privex-helpers[crypto]'    # Using standard pip3

# Or manually install the `cryptography` library.
pipenv install cryptography   # Using pipenv if you have it
pip3 install cryptography     # Using standard pip3
```

Summary

Some of the most useful parts of this module include `EncryptHelper` and `KeyManager` - these two components cover both symmetric (shared key) encryption/decryption, as well as asymmetric (public/private key) keypair generation, signing, verification, as well as encryption/decryption with RSA.

- `EncryptHelper` - Painless symmetric encryption / decryption with AES-128
- `KeyManager` - Painless generation of asymmetric keys, with signing/verification and en/decryption support

EncryptHelper - Painless symmetric encryption with AES-128

`EncryptHelper` is a wrapper class designed to make it extremely easy to use asymmetric encryption and decryption. At it's core is the `cryptography.fernet.Fernet` encryption system, and the class is designed to make usage of Fernet as painless as possible.

Basic Usage of EncryptHelper:

```
>>> from privex.helpers import EncryptHelper
>>> key = EncryptHelper.generate_key() # Generates a 32-byte symmetric key, returned␣
↪as a base64 encoded string
>>> key_out = EncryptHelper.generate_key('my_key.txt')  # Generates and saves a key␣
↪to my_key.txt, then returns it.
>>> crypt = EncryptHelper(key)          # Create an instance of EncryptHelper, en/
↪decrypting using ``key`` by default
# Encrypts the string 'hello world' with AES-128 CBC using the instance's key,␣
↪returned as a base64 string
>>> enc = crypt.encrypt_str('hello world')
>>> print(enc)
gAAAAABc7ERTpu2D_uven3l-KtU_ewUC8YWKqXEbLEKrPKrKWT138MNq-I9RRtCD8UZLdQrcdM_
↪IhUU6r8T16lQkoJZ-I7N39g==

>>> crypt.is_encrypted(enc)       # Check if a string/bytes is encrypted (only works␣
↪with data matching the key)
True
>>> data = crypt.decrypt_str(enc) # Decrypt the encrypted data using the same key,␣
↪outputs as a string
>>> print(data)
hello world
```

KeyManager - Painless generation of asymmetric keys, with signing/verification and en/decryption support

While the Cryptography library is a brilliant library with many features, and a good security track record - it's asymmetric key features require a ridiculous amount of scaffolding to make them usable in a project.

That's where `KeyManager` comes in.

Watch how simple it is to generate and save two types of asymmetric keys (RSA and Ed25519), and put them to use with signatures and encryption.

First, let's generate a keypair and save it to disk. Without any algorithm related configuration options, it will simply generate an RSA 2048 private and public key, and save them to the current working directory. For convenience, it will also return the private and public key, so you can make use of them immediately after generating them.

```
>>> rsa_priv, rsa_pub = KeyManager.output_keypair('id_rsa', 'id_rsa.pub')
```

Now, let's make an Ed25519 key - an algorithm quickly becoming common for SSH keys thanks to it's extremely small public + private keys that could practically fit in an SMS message.

```
>>> ed_priv, ed_pub = KeyManager.output_keypair('id_ed25519', 'id_ed25519.pub', alg=
↪'ed25519')
```

To use the signature / encryption functionality, first we have to load a key into `KeyManager`. There's two ways you can do this. The first is simply passing the key as a string/bytes into the constructor. The second is loading the key from disk (useful if you're using an existing key you have saved as a file).

Loading keys:

```
# Example 1. Let's just pass our Ed25519 private key bytes straight into KeyManager.
>>> km_ed = KeyManager(ed_priv)
# Example 2. For the RSA key, we'll load it from a file
>>> km_rsa = KeyManager.load_keyfile('id_rsa')
```

Now let's sign a message with each key, and then verify the message:

```
# Sign the message "hello world" with both our Ed25519 and RSA private key.
>>> msg = 'hello world'
>>> sig_ed = km_ed.sign(msg)
>>> sig_rsa = km_rsa.sign(msg)
# Now we can verify the signature using the public keys (automatically re-generated␣
↪from the private key)
>>> km_ed.verify(signature=sig_ed, message=msg)
True
>>> km_rsa.verify(signature=sig_rsa, message=msg)
True
# If the signature was invalid, e.g. if we pass the RSA signature to the Ed25519␣
↪KeyManager...
>>> km_ed.verify(sig_rsa, 'hello world')
    Traceback (most recent call last):
      File "<input>", line 1, in <module>
        raise InvalidSignature
    cryptography.exceptions.InvalidSignature
# So we can see that the signatures actually work :)
```

For RSA keys, we can also do encryption and decryption of small messages:

```
>>> enc = km_rsa.encrypt(msg)
# For easy storage/transmission, the encrypted data is base64 encoded. This ensures␣
↪you can transmit the
# encrypted message cleanly over email, HTTP etc. without the bytes getting garbled.
>>> enc
b'Sf1PC_TViZdA4lq7PwSnRTLbWX20vcCtkLyQWazE9EfM9_AIn6pNTHG...
# Now when we run `decrypt`, we get back our original message of "hello world"
```

(continues on next page)

(continued from previous page)

```
# Note: decrypt supports both raw bytes as well as base64 encoded data and will␣
↪automatically detect whether
# or not it has to decode base64.
>>> km_rsa.decrypt(enc)
b'hello world'
```

Copyright:

```
    +===================================================+
    |                 © 2019 Privex Inc.                |
    |               https://www.privex.io               |
    +===================================================+
    |                                                   |
    |        Originally Developed by Privex Inc.        |
    |        License: X11 / MIT                         |
    |                                                   |
    |        Core Developer(s):                         |
    |                                                   |
    |          (+)  Chris (@someguy123) [Privex]        |
    |          (+)  Kale (@kryogenic) [Privex]          |
    |                                                   |
    +===================================================+

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

3.10 privex.helpers.decorators

Class Method / Function decorators

Copyright:

```
    +===================================================+
    |                 © 2019 Privex Inc.                |
    |               https://www.privex.io               |
    +===================================================+
    |                                                   |
    |        Originally Developed by Privex Inc.        |
    |        License: X11 / MIT                         |
    |                                                   |
    |        Core Developer(s):                         |
    |                                                   |
    |          (+)  Chris (@someguy123) [Privex]        |
    |          (+)  Kale (@kryogenic) [Privex]          |
    |                                                   |
    +===================================================+

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

Functions

async_retry([max_retries, delay])	AsyncIO coroutine compatible version of *retry_on_err()* - for painless automatic retry-on-exception for async code.
mock_decorator(*dec_args, **dec_kwargs)	This decorator is a pass-through decorator which does nothing other than be a decorator.
r_cache(cache_key[, cache_time, ...])	This is a decorator which caches the result of the wrapped function with the global cache adapter from *privex.helpers.cache* using the key `cache_key` and with an expiry of `cache_time` seconds.
r_cache_async(cache_key[, cache_time, ...])	Async function/method compatible version of *r_cache()* - see docs for *r_cache()*
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.

3.10.1 async_retry

privex.helpers.decorators.**async_retry**(*max_retries: int = 3*, *delay: Union[int, float] = 3*, ***retry_conf*)

AsyncIO coroutine compatible version of *retry_on_err()* - for painless automatic retry-on-exception for async code.

Decorates an AsyncIO coroutine (`async def`) 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):

```
>>> from privex.helpers import async_retry
>>>
>>> @async_retry(5, 1, fail_on=[IOError])
... async def my_func(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:

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

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

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

We can also use `ignore_on` to "ignore" certain exceptions. Ignored exceptions cause the function to be retried with a delay, as normal, but without incrementing the total retries counter.

```
>>> from privex.helpers import async_retry
>>> import random
>>>
>>> @async_retry(5, 1, fail_on=[IOError], ignore=[ConnectionResetError])
... async def my_func(some=None, args=None):
...     if random.randint(1,10) > 7: raise ConnectionResetError()
...     if some == 'io': raise IOError()
...     raise FileExistsError()
...
```

To show this at work, we've enabled debug logging for you to see:

```
>>> await my_func()
[INFO]    <class 'ConnectionResetError'> -
[INFO]    Exception while running 'my_func', will retry 5 more times.
[DEBUG]   >> (?) Ignoring exception '<class 'ConnectionResetError'>' as exception␣
↪is in 'ignore' list.
          Ignore Count: 0 // Max Ignores: 100 // Instance Match: False

[INFO]    <class 'FileExistsError'> -
[INFO]    Exception while running 'my_func', will retry 5 more times.

[INFO]    <class 'ConnectionResetError'> -
[INFO]    Exception while running 'my_func', will retry 4 more times.
[DEBUG]   >> (?) Ignoring exception '<class 'ConnectionResetError'>' as exception␣
↪is in 'ignore' list.
          Ignore Count: 1 // Max Ignores: 100 // Instance Match: False

[INFO]    <class 'FileExistsError'> -
[INFO]    Exception while running 'my_func', will retry 4 more times.
```

As you can see above, when an ignored exception (`ConnectionResetError`) occurs, the remaining retry attempts doesn't go down. Instead, only the "Ignore Count" goes up.

Attention: For safety reasons, by default `max_ignore` is set to `100`. This means after 100 retries where an exception was ignored, the decorator will give up and raise the last exception.

This is to prevent the risk of infinite loops hanging your application. If you are 100% certain that the function you've wrapped, and/or the exceptions passed in `ignore` cannot cause an infinite retry loop, then you can pass `max_ignore=False` to the decorator to disable failure after `max_ignore` ignored exceptions.

Parameters

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

Key list fail_on A list() of Exception types that should result in immediate failure (don't retry, raise)

Key list ignore A list() of Exception types that should be ignored (will retry, but without incrementing the failure counter)

Key int|bool max_ignore (Default: `100`) If an exception is raised while retrying, and more than this many exceptions (listed in `ignore`) have been ignored during retry attempts, then give up and raise the last exception.

This feature is designed to prevent "ignored" exceptions causing an infinite retry loop. By default `max_ignore` is set to `100`, but you can increase/decrease this as needed.

You can also set it to `False` to disable raising when too many exceptions are ignored - however, it's strongly not recommended to disable `max_ignore`, especially if you have `instance_match=True`, as it could cause an infinite retry loop which hangs your application.

Key bool instance_match (Default: `False`) If this is set to `True`, then the exception type comparisons for `fail_on` and `ignore` will compare using `isinstance(e, x)` instead of `type(e) is x`.

If this is enabled, then exceptions listed in `fail_on` and `ignore` will also **match sub-classes** of the listed exceptions, instead of exact matches.

Key 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

Key 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.

3.10.2 mock_decorator

`privex.helpers.decorators.`**`mock_decorator`**(**dec_args*, ***dec_kwargs*)

This decorator is a pass-through decorator which does nothing other than be a decorator.

It's designed to be used with the `privex.helpers.common.Mocker` class when mocking classes/modules, allowing you to add fake decorators to the mock class/method which do nothing, other than act like a decorator without breaking your functions/methods.

3.10.3 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 with the global cache adapter from *`privex.helpers.cache`* 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 cache 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 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 cache 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'])
```

(continues on next page)

(continued from previous page)

```
... 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␣
↪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␣
↪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␣
↪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␣
↪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

- **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 cache 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)
- **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.

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

3.10.4 r_cache_async

privex.helpers.decorators.**r_cache_async**(*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

Async function/method compatible version of *r_cache()* - see docs for *r_cache()*

Basic usage:

```
>>> from privex.helpers import r_cache_async
>>> @r_cache_async('my_cache_key')
>>> async def some_func(some: int, args: int = 2):
...     return some + args
>>> await some_func(5, 10)
15

>>> # If we await some_func a second time, we'll get '15' again because it was␣
↪cached.
>>> await some_func(2, 3)
15
```

Async `cache_key` generation (you can also use normal synchronous functions/lambdas):

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

Parameters

- **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 cache 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)
- **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.

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

3.10.5 retry_on_err

privex.helpers.decorators.**retry_on_err**(*max_retries: int = 3*, *delay: Union[int, float] = 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')
```

Attention: For safety reasons, by default `max_ignore` is set to `100`. This means after 100 retries where an exception was ignored, the decorator will give up and raise the last exception.

This is to prevent the risk of infinite loops hanging your application. If you are 100% certain that the function you've wrapped, and/or the exceptions passed in `ignore` cannot cause an infinite retry loop, then you can pass `max_ignore=False` to the decorator to disable failure after `max_ignore` ignored exceptions.

Parameters

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

Key list fail_on A list() of Exception types that should result in immediate failure (don't retry, raise)

Key list ignore A list() of Exception types that should be ignored (will retry, but without incrementing the failure counter)

Key int|bool max_ignore (Default: `100`) If an exception is raised while retrying, and more than this many exceptions (listed in `ignore`) have been ignored during retry attempts, then give up and raise the last exception.

This feature is designed to prevent "ignored" exceptions causing an infinite retry loop. By default `max_ignore` is set to `100`, but you can increase/decrease this as needed.

You can also set it to `False` to disable raising when too many exceptions are ignored - however, it's strongly not recommended to disable `max_ignore`, especially if you have `instance_match=True`, as it could cause an infinite retry loop which hangs your application.

Key bool instance_match (Default: `False`) If this is set to `True`, then the exception type comparisons for `fail_on` and `ignore` will compare using `isinstance(e, x)` instead of `type(e) is x`.

If this is enabled, then exceptions listed in `fail_on` and `ignore` will also **match sub-classes** of the listed exceptions, instead of exact matches.

Key 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

Key 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.

Classes

FO	alias of *privex.helpers.decorators.FormatOpt*
FormatOpt(value)	This enum represents various options available for *r_cache()* 's `format_opt` parameter.

3.10.6 FO

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

3.10.6.1 Attributes

Attributes

KWARG_ONLY
MIX
POS_AUTO
POS_ONLY

3.10.6.1.1 KWARG_ONLY

FO.**KWARG_ONLY = 'kwarg'**

3.10.6.1.2 MIX

FO.**MIX = 'mix'**

3.10.6.1.3 POS_AUTO

FO.**POS_AUTO = 'force_pos'**

3.10.6.1.4 POS_ONLY

FO.**POS_ONLY = 'pos_only'**

3.10.7 FormatOpt

class privex.helpers.decorators.**FormatOpt**(*value*)

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.

3.10.7.1 Attributes

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.
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.

3.10.7.1.1 KWARG_ONLY

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

3.10.7.1.2 MIX

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

3.10.7.1.3 POS_AUTO

`FormatOpt.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')

3.10.7.1.4 POS_ONLY

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

3.11 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.        |
    |        License: X11 / MIT                         |
    |                                                   |
    |        Core Developer(s):                         |
    |                                                   |
    |          (+)  Chris (@someguy123) [Privex]        |
```

(continues on next page)

(continued from previous page)

```
    |          (+)  Kale (@kryogenic) [Privex]          |
    |                                                   |
    +===================================================+

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

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

3.11.1 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)

3.11.2 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):
```

(continues on next page)

(continued from previous page)

```
>>>     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.

3.11.3 model_to_dict

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

1 dimensional json-ifyer for any Model

3.11.4 to_json

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

Iterate a Django query set and dump to json str

3.12 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.               |
    |        License: X11 / MIT                                |
    |                                                          |
    |        Core Developer(s):                                |
    |                                                          |
    |          (+)  Chris (@someguy123) [Privex]               |
    |          (+)  Kale (@kryogenic) [Privex]                 |
    |                                                          |
    +==========================================================+

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

Exceptions

`BaseDNSException`	Base exception for DNS-related exceptions
`BoundaryException`	Thrown when the v4/v6 address boundary for a reverse DNS record is invalid.
`CacheNotFound`	Thrown when a cache key is requested, but it doesn't exist / is expired.
`DomainNotFound`	Thrown when a (sub)domain or it's parent(s) could not be found
`EncryptKeyMissing`	Raised when ENCRYPT_KEY is not set, or invalid
`EncryptionError`	Raised when something went wrong attempting to encrypt or decrypt a piece of data
`GeoIPAddressNotFound`	
`GeoIPDatabaseNotFound`	
`GeoIPException`	
`InvalidDNSRecord`	Thrown when a passed DNS record is not valid
`InvalidFormat`	Raised when an invalid public/private format, or encoding is specified when serializing an asymmetric key pair
`InvalidHost`	Raised when a passed IP address or hostname/domain is invalid.
`NetworkUnreachable`	Thrown when a network interface or IP version (e.g.
`NotConfigured`	Thrown when code attempts to access something that wasn't fully configured / instantiated by the user.
`NotFound`	Generic exception mixin for all exceptions related to something not being found
`PrivexException`	Base exception for all custom Privex exceptions
`ReverseDNSNotFound`	Raised when a given IP address does not have a reverse DNS set
`SysCallError`	Raised when an error appears to have been returned after calling an external command (e.g.

3.13 privex.helpers.extras

Various helper functions/classes which depend on a certain package being installed.

This constructor file attempts to load each extras module individually, each wrapped with a try/catch for `ImportError` so that one unavailable package doesn't cause problems.

3.14 privex.helpers.net

Network related helper code

Copyright:

```
    +===================================================+
    |                 © 2019 Privex Inc.                |
    |               https://www.privex.io               |
    +===================================================+
    |                                                   |
    |        Originally Developed by Privex Inc.        |
```

(continues on next page)

(continued from previous page)

```
    |        License: X11 / MIT                          |
    |                                                     |
    |        Core Developer(s):                           |
    |                                                     |
    |          (+)  Chris (@someguy123) [Privex]          |
    |          (+)  Kale (@kryogenic) [Privex]            |
    |                                                     |
    +=====================================================+

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

Functions

asn_to_name(as_number[, quiet])	Look up an integer Autonomous System Number and return the human readable name of the organization.
get_rdns(host[, throw])	Look up the reverse DNS hostname for `host` and return it as a string.
get_rdns_multi(*hosts[, throw])	Resolve reverse DNS hostnames for multiple IPs / domains specified as positional arguments.
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
ping(ip[, timeout])	Sends a ping to a given IPv4 / IPv6 address.
resolve_ip(addr[, version, v4_convert])	Wrapper for *resolve_ips()* - passes args to *resolve_ips()* and returns the first item from the results.
resolve_ips(addr[, version, v4_convert])	With just a single hostname argument, both IPv4 and IPv6 addresses will be returned as strings.
resolve_ips_multi(*addr[, version, v4_convert])	Resolve IPv4/v6 addresses for multiple hosts specified as positional arguments.

3.14.1 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'

3.14.2 get_rdns

privex.helpers.net.**get_rdns**(*host: Union[str, ipaddress.IPv4Address, ipaddress.IPv6Address]*, *throw=True*) → Optional[str]

Look up the reverse DNS hostname for `host` and return it as a string. The `host` can be an IP address as a `str`, `IPv4Address`, `IPv6Address` - or a domain.

If a domain is passed, e.g. `privex.io` - then the reverse DNS will be looked up for the IP address contained in the domain's AAAA or A records.

Toggle `throw` to control whether to raise exceptions on error (`True`), or to simply return None (`False`).

Basic usage:

```
>>> from privex.helpers import get_rdns
>>> get_rdns('185.130.44.10')
'web-se1.privex.io'
>>> get_rdns('2a07:e00::333')
'se.dns.privex.io'
>>> get_rdns('privex.io')
'web-se1.privex.io'
```

Error handling:

```
>>> get_rdns('192.168.4.5')
Traceback (most recent call last):
  File "<ipython-input-14-e1ed65295031>", line 1, in <module>
    get_rdns('192.168.4.5')
privex.helpers.exceptions.ReverseDNSNotFound: No reverse DNS records found for␣
↪host '192.168.4.5':
    <class 'socket.herror'> [Errno 1] Unknown host
>>> get_rdns('non-existent-domain.example')
Traceback (most recent call last):
  File "<ipython-input-16-0d75d37a930f>", line 1, in <module>
    get_rdns('non-existent-domain.example')
privex.helpers.exceptions.InvalidHost: Host 'non-existent-domain.example' is not␣
↪a valid IP address,
nor an existent domain: <class 'socket.gaierror'> [Errno 8] nodename nor servname␣
↪provided, or not known
>>> repr(get_rdns('192.168.4.5', throw=False))
'None'
>>> repr(get_rdns('non-existent-domain.example', False))
'None'
```

Parameters

- **host** (*str | IPv4Address | IPv6Address*) – An IPv4/v6 address, or domain to lookup reverse DNS for.
- **throw** (*bool*) – (Default: `True`) When `True`, will raise `ReverseDNSNotFound` or `InvalidHost` when no rDNS records can be found for `host`, or when `host` is an invalid IP / non-existent domain. When `False`, will simply return `None` when `host` is invalid, or no rDNS records are found.

Raises

- **ReverseDNSNotFound** – When `throw` is True and no rDNS records were found for `host`
- **InvalidHost** – When `throw` is True and `host` is an invalid IP address or non-existent domain/hostname

Return Optional[str] rDNS The reverse DNS hostname for `host` (value of PTR record)

3.14.3 get_rdns_multi

privex.helpers.net.**get_rdns_multi**(**hosts: Union[str, ipaddress.IPv4Address, ipaddress.IPv6Address], throw=False*) → Generator[Tuple[str, Optional[str]], None, None]

Resolve reverse DNS hostnames for multiple IPs / domains specified as positional arguments.

Each host in `hosts` can be an IP address as a `str`, `IPv4Address`, `IPv6Address` - or a domain.

Returns results as a generator, to allow for efficient handling of a large amount of hosts to resolve.

Basic usage:

```
>>> for host, rdns in get_rdns_multi('185.130.44.10', '8.8.4.4', '1.1.1.1',
↪'2a07:e00::333'):
>>>     print(f"{host:<20} -> {rdns:>5}")
185.130.44.10        -> web-se1.privex.io
8.8.4.4              -> dns.google
1.1.1.1              -> one.one.one.one
2a07:e00::333        -> se.dns.privex.io
```

If you're only resolving a small number of hosts (less than 100 or so), you can simply cast the generator into a `dict` using `dict()`, which will get you a dictionary of hosts mapped to their rDNS:

```
>>> data = dict(get_rdns_multi('185.130.44.10', '8.8.4.4', '1.1.1.1',
↪'2a07:e00::333'))
>>> data['8.8.4.4']
'dns.google'
>>> data.get('2a07:e00::333', 'error')
'se.dns.privex.io'
```

Parameters

- **hosts** (*str | IPv4Address | IPv6Address*) – One or more IPv4/v6 addresses, or domains to lookup reverse DNS for - as positional args.
- **throw** (*bool*) – (Default: `False`) When `True`, will raise `ReverseDNSNotFound` or `InvalidHost` when no rDNS records can be found for a host, or when the host is an invalid IP / non-existent domain. When `False`, will simply return `None` when a host is invalid, or no rDNS records are found.

Raises

- **ReverseDNSNotFound** – When `throw` is True and no rDNS records were found for `host`
- **InvalidHost** – When `throw` is True and `host` is an invalid IP address or non-existent domain/hostname

Return Tuple[str,Optional[str]] rDNS A generator returning `tuple`'s containing the original passed host, and it's reverse DNS hostname (value of PTR record)

3.14.4 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`

3.14.5 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
- **v6_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`

3.14.6 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)

3.14.7 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)

3.14.8 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.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`

3.14.9 ping

privex.helpers.net.**ping**(*ip: str*, *timeout: int = 30*) → bool

Sends a ping to a given IPv4 / IPv6 address. Tested with IPv4+IPv6 using `iputils-ping` on Linux, as well as the default IPv4 `ping` utility on Mac OSX (Mojave, 10.14.6).

Fully supported when using Linux with the `iputils-ping` package. Only IPv4 support on Mac OSX.

Example Usage:

```
>>> from privex.helpers import ping
>>> if ping('127.0.0.1', 5) and ping('::1', 10):
...     print('Both 127.0.0.1 and ::1 are up')
... else:
...     print('127.0.0.1 or ::1 failed to respond to a ping within the given␣
↪timeout.')
```

Known Incompatibilities:

- NOT compatible with IPv6 addresses on OSX due to the lack of a timeout argument with `ping6`
- NOT compatible with IPv6 addresses when using `inetutils-ping` on Linux due to separate `ping6` command

Parameters

- **ip** (*str*) – An IP address as a string, e.g. `192.168.1.1` or `2a07:e00::1`
- **timeout** (*int*) – (Default: 30) Number of seconds to wait for a response from the ping before timing out

Raises **ValueError** – When the given IP address `ip` is invalid or `timeout` < 1

Return bool `True` if ping got a response from the given IP, `False` if not

3.14.10 resolve_ip

privex.helpers.net.**resolve_ip**(*addr: Union[str, ipaddress.IPv4Address, ipaddress.IPv6Address]*, *version: Union[str, int] = 'any'*, *v4_convert=False*) → Optional[str]

Wrapper for *resolve_ips()* - passes args to *resolve_ips()* and returns the first item from the results.

If the results are empty, `None` will be returned.

Examples:

```
>>> resolve_ip('privex.io')
'2a07:e00::abc'
>>> resolve_ip('privex.io', 'v4')
'185.130.44.10'
>>> resolve_ip('microsoft.com')
'104.215.148.63'
>>> repr(resolve_ip('microsoft.com' ,'v6'))
'None'
>>> resolve_ip('microsoft.com' ,'v6', v4_convert=True)
'::ffff:104.215.148.63'
```

Parameters

- **addr** (*str | IPv4Address | IPv6Address*) – Hostname to resolve / IP address to validate or pass-thru
- **version** (*str | int*) – (Default: `any`) - `4` (int), `'v4'`, `6` (int), `'v6'` (see *resolve_ips()* for more options)
- **v4_convert** (*bool*) – (Default: `False`) If set to `True`, will allow IPv6-wrapped IPv4 addresses starting with `::ffff:` to be returned when requesting version `v6` from an IPv4-only hostname.

Raises **AttributeError** – Raised when an IPv4 address is passed and `version` is set to IPv6 - as well as vice versa (IPv6 passed while version is set to IPv4)

Return Optional[str] ips An IPv4/v6 address as a string if there was at least 1 result - otherwise `None`.

3.14.11 resolve_ips

privex.helpers.net.**resolve_ips**(*addr: Union[str, ipaddress.IPv4Address, ipaddress.IPv6Address]*, *version: Union[str, int] = 'any'*, *v4_convert=False*) → List[str]

With just a single hostname argument, both IPv4 and IPv6 addresses will be returned as strings:

```
>>> resolve_ips('www.privex.io')
['2a07:e00::abc', '185.130.44.10']
```

You can provide the `version` argument as either positional or kwarg, e.g. `v4` or `v6` to restrict the results to only that IP version:

```
>>> resolve_ips('privex.io', version='v4')
['185.130.44.10']
```

The `v4_convert` option is `False` by default, which prevents `::ffff:` style IPv6 wrapped IPv4 addresses being returned when you request version `v6`:

```
>>> resolve_ips('microsoft.com')
['40.76.4.15', '40.112.72.205', '13.77.161.179', '40.113.200.201', '104.215.148.63
↪']
>>> resolve_ips('microsoft.com', 'v6')
[]
```

If for whatever reason, you need `::ffff:` IPv6 wrapped IPv4 addresses to be returned, you can set `v4_convert=True`, which will disable filtering out `::ffff:` fake IPv6 addresses:

```
>>> resolve_ips('microsoft.com', 'v6', v4_convert=True)
['::ffff:40.76.4.15', '::ffff:40.112.72.205', '::ffff:13.77.161.179',
 '::ffff:40.113.200.201', '::ffff:104.215.148.63']
```

For convenience, if an IPv4 / IPv6 address is specified, then it will simply be validated against `version` and then returned within a list. This is useful when handling user specified data, which may be either a hostname or an IP address, and you need to resolve hostnames while leaving IP addresses alone:

```
>>> resolve_ips('8.8.4.4')
['8.8.4.4']
>>> resolve_ips('2a07:e00::333')
['2a07:e00::333']
```

(continues on next page)

(continued from previous page)

```
>>> resolve_ips('8.8.4.4', 'v6')
Traceback (most recent call last):
  File "<ipython-input-10-6ca9e766006f>", line 1, in <module>
    resolve_ips('8.8.4.4', 'v6')
AttributeError: Passed address '8.8.4.4' was an IPv4 address, but 'version'␣
↪requested an IPv6 address.

>>> resolve_ips('2a07:e00::333', 'v4')
Traceback (most recent call last):
  File "<ipython-input-11-543bfa71c57a>", line 1, in <module>
    resolve_ips('2a07:e00::333', 'v4')
AttributeError: Passed address '2a07:e00::333' was an IPv6 address, but 'version'␣
↪requested an IPv4 address.
```

Parameters

- **addr** (*str | IPv4Address | IPv6Address*) – The hostname to resolve. If an IPv4 / IPv6 address is passed instead of a hostname, it will be validated against `version`, then returned in a single item list.
- **version** (*str | int*) – Default: `'any'` - Return both IPv4 and IPv6 addresses (if both are found). If an IP address is passed, then both IPv4 and IPv6 addresses will be accepted. If set to one of the IPv4/IPv6 version choices, then a passed IP of the wrong version will raise `AttributeError`

  Choices:

  - **IPv4 Options**: `4` (int), `'v4'`, `'4'` (str), `'ipv4'`, `'inet'`, `'inet4'`
  - **IPv6 Options**: `6` (int), `'v6'`, `'6'` (str), `'ipv6'`, `'inet6'`
- **v4_convert** (*bool*) – (Default: `False`) If set to `True`, will allow IPv6-wrapped IPv4 addresses starting with `::ffff:` to be returned when requesting version `v6` from an IPv4-only hostname.

Raises **AttributeError** – Raised when an IPv4 address is passed and `version` is set to IPv6 - as well as vice versa (IPv6 passed while version is set to IPv4)

Return List[str] ips Zero or more IP addresses in a list of `str`'s

3.14.12 resolve_ips_multi

privex.helpers.net.**resolve_ips_multi**(**addr: Union[str, ipaddress.IPv4Address, ipaddress.IPv6Address], version: Union[str, int] = 'any', v4_convert=False*) → Generator[Tuple[str, Optional[List[str]]], None, None]

Resolve IPv4/v6 addresses for multiple hosts specified as positional arguments.

Returns results as a generator, to allow for efficient handling of a large amount of hostnames to resolve.

Using the generator in a loop efficiently:

```
>>> for host, ips in resolve_ips_multi('privex.io', 'cloudflare.com', 'google.com
↪'):
...     print(f"{host:<20} ->   {', '.join(ips)}")
...
privex.io            ->   2a07:e00::abc, 185.130.44.10
```

(continues on next page)

(continued from previous page)

```
cloudflare.com       ->   2606:4700::6811:af55, 2606:4700::6811:b055, 104.17.176.
↪85, 104.17.175.85
google.com           ->   2a00:1450:4009:807::200e, 216.58.204.238
```

If you're only resolving a small number of hosts (less than 100 or so), you can simply cast the generator into a `dict` using `dict()`, which will get you a dictionary of hosts mapped to lists of IP addresses.

Dictionary Cast Examples:

```
>>> dict(resolve_ips_multi('privex.io', 'microsoft.com', 'google.com'))
{'privex.io': ['2a07:e00::abc', '185.130.44.10'],
 'microsoft.com': ['104.215.148.63', '40.76.4.15', '40.112.72.205', '40.113.200.
↪201', '13.77.161.179'],
 'google.com': ['2a00:1450:4009:807::200e', '216.58.204.238']}
>>> dict(resolve_ips_multi('privex.io', 'microsoft.com', 'google.com', version='v6
↪'))
{'privex.io': ['2a07:e00::abc'], 'microsoft.com': [], 'google.com': [
↪'2a00:1450:4009:81c::200e']}
>>> dict(resolve_ips_multi('privex.io', 'this-does-not-exist', 'google.com',␣
↪version='v6'))
{'privex.io': ['2a07:e00::abc'], 'this-does-not-exist': [], 'google.com': [
↪'2a00:1450:4009:81c::200e']}
 >>> dict(resolve_ips_multi('privex.io', 'example.com', '127.0.0.1', version='v6
↪'))
[resolve_ips_multi AttributeError] Invalid IP: 127.0.0.1 - Ex: <class
↪'AttributeError'> Passed address '127.0.0.1' was
                                  an IPv4 address, but 'version' requested an␣
↪IPv6 address.
{'privex.io': ['2a07:e00::abc'], 'example.com': [
↪'2606:2800:220:1:248:1893:25c8:1946'], '127.0.0.1': None}
```

Parameters

- **addr** (*str|IPv4Address|IPv6Address*) – Hostname to resolve / IP address to validate or pass-thru
- **version** (*str|int*) – (Default: `any`) - `4` (int), `'v4'`, `6` (int), `'v6'` (see *resolve_ips()* for more options)
- **v4_convert** (*bool*) – (Default: `False`) If set to `True`, will allow IPv6-wrapped IPv4 addresses starting with `::ffff:` to be returned when requesting version `v6` from an IPv4-only hostname.

Return Tuple[str,Optional[List[str]] gen A generator which returns tuples containing a hostname/IP, and a list of it's resolved IPs. If the IP was rejected (e.g. IPv4 IP passed with `v6` `version` param), then the list may instead be `None`.

3.15 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:

Attributes

—

Functions

clean_threadstore([thread_id, name])	Remove the per-thread instance storage in __STORE, usually called when a thread is exiting.
close_geoip(geo_type[, thread_id])	Close the global GeoIP connection and delete the instance.
close_memcached_async([thread_id])	Close the global Async Memcached connection and delete the instance.
close_redis([thread_id])	Close the global Redis connection and delete the instance.
close_redis_async([thread_id])	Close the global Async Redis connection and delete the instance.
configure_memcached_async([host, port])	Update global Memcached settings and re-instantiate the global Async Redis instance with the new settings.
configure_redis([host, port, db])	Update global Redis settings and re-instantiate the global Redis instance with the new settings.
configure_redis_async([host, port, db])	Update global Redis settings and re-instantiate the global Async Redis instance with the new settings.

continues on next page

Table 32 – continued from previous page

connect_geoip(*args, **geo_config)	
connect_memcached_async(**rd_config)	
connect_redis(*args, **rd_config)	
connect_redis_async(**rd_config)	
get_geodbs()	
get_geoip(geo_type[, new_connection, thread_id])	Get a GeoIP Reader object.
get_geoip_db(geo_type)	Return the full path to the GeoIP2 database for `geo_type`.
get_memcached_async([new_connection, thread_id])	Get an Async Memcached connection object.
get_redis([new_connection, thread_id])	Get a Redis connection object.
get_redis_async([new_connection, thread_id])	Get an Async Redis connection object.
reset_geoip(geo_type[, thread_id])	Close the global GeoIP connection, delete the instance, then re-instantiate it
reset_memcached_async([thread_id])	Close the global Async Memcached connection, delete the instance, then re-instantiate it
reset_redis([thread_id])	Close the global Redis connection, delete the instance, then re-instantiate it
reset_redis_async([thread_id])	Close the global Async Redis connection, delete the instance, then re-instantiate it

3.15.1 clean_threadstore

privex.helpers.plugin.**clean_threadstore**(*thread_id=None*, *name=None*)

Remove the per-thread instance storage in ___STORE, usually called when a thread is exiting.

Example:

```
>>> def some_thread():
...     r = get_redis()
...     print('doing something')
...     print('cleaning up...')
...     clean_threadstore()       # With no arguments, it cleans the thread store␣
↪for the thread that called it.
>>> t = threading.Thread(target=some_thread)
>>> t.start()
>>> t.join()
```

Usage outside of a thread:

```
>>> t = threading.Thread(target=some_thread)
>>> t.start()
>>> thread_id = t.ident                      # Get the thread ID for the started␣
↪thread
>>> t.join()                                 # Wait for the thread to finish
>>> if thread_id is not None:                # Make sure the thread ID isn't None
...     clean_threadstore(thread_id)         # Cleanup any leftover instances, if␣
↪there are any.
...
```

Removing an individual item from thread store:

```
>>> def some_thread():
...     r = get_redis()
...     print('doing something')
...     print('cleaning up...')
...     clean_threadstore(name='redis')    # Delete only the key 'redis' from the␣
↪thread store
```

Parameters

- **thread_id** – The ID of the thread (usually from `threading.get_ident()`) to clean the storage for. If left as None, will use the ID returned by `threading.get_ident()`.
- **name** – If specified, then only the key `name` will be deleted from the thread store, instead of the entire thread store.

3.15.2 close_geoip

`privex.helpers.plugin.`**`close_geoip`**(*geo_type: str*, *thread_id=None*) → bool

Close the global GeoIP connection and delete the instance.

Parameters

- **geo_type** (*str*) – The GeoIP database type: either 'city', 'asn' or 'country'
- **thread_id** – Close and delete the Redis instance for this thread ID, instead of the detected current thread

Return bool deleted `True` if an instance was found and deleted. `False` if there was no existing Redis instance.

3.15.3 close_memcached_async

`privex.helpers.plugin.`**`close_memcached_async`**(*thread_id=None*) → bool

Close the global Async Memcached connection and delete the instance.

Parameters **thread_id** – Close and delete the Memcached instance for this thread ID, instead of the detected current thread

Return bool deleted `True` if an instance was found and deleted. `False` if there was no existing Memcached instance.

3.15.4 close_redis

`privex.helpers.plugin.`**`close_redis`**(*thread_id=None*) → bool

Close the global Redis connection and delete the instance.

Parameters **thread_id** – Close and delete the Redis instance for this thread ID, instead of the detected current thread

Return bool deleted `True` if an instance was found and deleted. `False` if there was no existing Redis instance.

3.15.5 close_redis_async

async privex.helpers.plugin.**close_redis_async**(*thread_id=None*) → bool

Close the global Async Redis connection and delete the instance.

Parameters **thread_id** – Close and delete the Redis instance for this thread ID, instead of the detected current thread

Return bool deleted `True` if an instance was found and deleted. `False` if there was no existing Redis instance.

3.15.6 configure_memcached_async

async privex.helpers.plugin.**configure_memcached_async**(*host='localhost'*, *port: int = 11211*, ***kwargs*)

Update global Memcached settings and re-instantiate the global Async Redis instance with the new settings.

3.15.7 configure_redis

privex.helpers.plugin.**configure_redis**(*host='localhost'*, *port: int = 6379*, *db: int = 0*, ***kwargs*)

Update global Redis settings and re-instantiate the global Redis instance with the new settings.

3.15.8 configure_redis_async

async privex.helpers.plugin.**configure_redis_async**(*host='localhost'*, *port: int = 6379*, *db: int = 0*, ***kwargs*)

Update global Redis settings and re-instantiate the global Async Redis instance with the new settings.

3.15.9 connect_geoip

privex.helpers.plugin.**connect_geoip**(**args*, ***geo_config*) → geoip2.database.Reader

3.15.10 connect_memcached_async

async privex.helpers.plugin.**connect_memcached_async**(***rd_config*) → aiomcache.client.Client

3.15.11 connect_redis

privex.helpers.plugin.**connect_redis**(**args*, ***rd_config*)

3.15.12 connect_redis_async

async privex.helpers.plugin.**connect_redis_async**(***rd_config*) → aioredis.commands.Redis

3.15.13 get_geodbs

privex.helpers.plugin.**get_geodbs**() → *privex.helpers.collections.DictObject*

3.15.14 get_geoip

privex.helpers.plugin.**get_geoip**(*geo_type: str*, *new_connection=False*, *thread_id=None*, ***geo_config*) → geoip2.database.Reader

Get a GeoIP Reader object. Create one if it doesn't exist.

3.15.15 get_geoip_db

privex.helpers.plugin.**get_geoip_db**(*geo_type: str*) → str

Return the full path to the GeoIP2 database for `geo_type`.

If we haven't yet scanned the search paths for the database, then `_find_geoip()` will be called to try and locate the database file.

If the database is found, the `_DETECTED` boolean setting will be changed to `True` so we know that the path contained in the *`get_geodbs()`* result is valid in the future, avoiding unnecessary searches.

If the database can't be found anywhere within the search paths, `GeoIPDatabaseNotFound` will be raised.

Parameters **geo_type** (*str*) – The GeoIP database type: either 'city', 'asn' or 'country'

Raises **GeoIPDatabaseNotFound** – If the database for `geo_type` could not be found.

Return str path The full path to the detected GeoIP database

3.15.16 get_memcached_async

async privex.helpers.plugin.**get_memcached_async**(*new_connection=False*, *thread_id=None*, ***rd_config*) → aiomcache.client.Client

Get an Async Memcached connection object. Create one if it doesn't exist.

3.15.17 get_redis

privex.helpers.plugin.**get_redis**(*new_connection=False*, *thread_id=None*, ***rd_config*) → redis.client.Redis

Get a Redis connection object. Create one if it doesn't exist.

3.15.18 get_redis_async

async privex.helpers.plugin.**get_redis_async**(*new_connection=False*, *thread_id=None*, ***rd_config*) → <module 'aioredis.connection' from '/home/docs/checkouts/readthedocs.org/user_builds/python-helpers/envs/develop/lib/python3.7/site-packages/aioredis/connection.py'>

Get an Async Redis connection object. Create one if it doesn't exist.

Usage:

```
>>> redis_conn = await get_redis_async()
>>> redis = await redis_conn
>>> await redis.set('some_key', 'example')
>>> await redis.get('some_key')
'example'
```

3.15.19 reset_geoip

privex.helpers.plugin.**reset_geoip**(*geo_type: str*, *thread_id=None*) → geoip2.database.Reader

Close the global GeoIP connection, delete the instance, then re-instantiate it

3.15.20 reset_memcached_async

async privex.helpers.plugin.**reset_memcached_async**(*thread_id=None*) → aiomcache.client.Client

Close the global Async Memcached connection, delete the instance, then re-instantiate it

3.15.21 reset_redis

privex.helpers.plugin.**reset_redis**(*thread_id=None*) → redis.client.Redis

Close the global Redis connection, delete the instance, then re-instantiate it

3.15.22 reset_redis_async

async privex.helpers.plugin.**reset_redis_async**(*thread_id=None*) → <module 'aioredis.connection' from '/home/docs/checkouts/readthedocs.org/user_builds/python-helpers/envs/develop/lib/python3.7/site-packages/aioredis/connection.py'>

Close the global Async Redis connection, delete the instance, then re-instantiate it

3.16 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.       |
    |        License: X11 / MIT                        |
    |                                                  |
    |        Core Developer(s):                        |
    |                                                  |
    |          (+)  Chris (@someguy123) [Privex]       |
    |          (+)  Kale (@kryogenic) [Privex]         |
    |                                                  |
    +==================================================+

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

Attributes

—

3.17 privex.helpers.setuppy

Helpers for setup.py, e.g. requirements.txt parsing, version bumping, custom setup.py commands

Inside of `privex.helpers.setuppy.common` there's a variety of functions related to generating requirements.txt files, parsing requirements.txt files which recursively import other requirements.txt files, and handing automatic generation of `extras_require` from a folder containing requirements txt files.

Inside of `privex.helpers.setuppy.bump` - most notably is `bump_version()` - a function which detects a package's version, increments the appropriate part of the version number, and then updates the python file containing the version number (e.g. an `__init__.py`)

Inside of `privex.helpers.setuppy.commands` there are command classes which can be loaded into setup.py to assist with building python packages, generating requirements.txt files from extras, as well as general management such as a `BumpCommand` which allows you to bump your package version with a simple `./setup.py bump --minor`

More detailed usage documentation is available within each individual module's documentation.

3.18 privex.helpers.types

Attributes

—

Classes

NO_RESULT	Simple functionless type which means "no results were found or nothing matched this function's query".
USE_ORIG_VAR	A simple functionless type, used purely as a default parameter value meaning "fallback to the value from a certain other parameter".

3.18.1 NO_RESULT

privex.helpers.types.**NO_RESULT**

Simple functionless type which means "no results were found or nothing matched this function's query".

Useful for returning a unique "nothing to return" value from functions where `None` / `False` might be considered as successful, and exceptions aren't suitable:

```
>>> from privex.helpers.types import NO_RESULT
>>> def some_func(x: int):
...     if (x + 1) > 2: return True
...     elif (x + 1) < 2: return False
...     if x == 0 or (x + 1) == 0: return None
...     return NO_RESULT
>>> res = some_func(-2)
>>> res == NO_RESULT
True
```

alias of `privex.helpers.types.NoResult`

3.18.1.1 Methods

Methods

—

3.18.2 USE_ORIG_VAR

privex.helpers.types.**USE_ORIG_VAR**

A simple functionless type, used purely as a default parameter value meaning "fallback to the value from a certain other parameter".

Primarily used in *empty_if()* but can be used by any function/method, including use outside of privex-helpers.

alias of `privex.helpers.types.UseOrigVar`

3.18.2.1 Methods

Methods

—

3.19 How to use the unit tests

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

3.19.1 Testing pre-requisites

- Ensure you have any mandatory requirements installed (see setup.py's install_requires)
- You should install `pytest` to run the tests, it works much better than standard python unittest.
- 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.

For the best testing experience, it's recommended to install the `dev` extra, which includes every optional dependency, as well as development requirements such as `pytest` , `coverage` as well as requirements for building the documentation.

3.19.2 Running via PyTest

To run the tests, we strongly recommend using the `pytest` tool (used by default for our Travis CI):

```
# Install PyTest if you don't already have it.
user@host: ~/privex-helpers $ pip3 install pytest

# We recommend adding the option ``-rxXs`` which will show information about why␣
↪certain tests were skipped
# as well as info on xpass / xfail tests
# You can add `-v` for more detailed output, just like when running the tests␣
↪directly.
user@host: ~/privex-helpers $ pytest -rxXs

# NOTE: If you're using a virtualenv, sometimes you may encounter strange conflicts␣
↪between a global install
# of PyTest, and the virtualenv PyTest, resulting in errors related to packages not␣
↪being installed.
# A simple workaround is just to call pytest as a module from the python3 executable:

user@host: ~/privex-helpers $ python3 -m pytest -rxXs

============================= test session starts ==============================
platform darwin -- Python 3.7.0, pytest-5.2.2, py-1.8.0, pluggy-0.13.0
rootdir: /home/user/privex-helpers
collected 99 items

tests/test_bool.py .........                                             [  9%]
tests/test_cache.py .................                                    [ 25%]
tests/test_crypto.py ........................                            [ 50%]
```

(continues on next page)

(continued from previous page)

```
tests/test_general.py ...................                                [ 69%]
tests/test_net.py ssss.s                                                 [ 75%]
tests/test_parse.py ..........                                           [ 85%]
tests/test_rdns.py ..............                                        [100%]

============================ short test summary info ============================
SKIPPED [1] tests/test_net.py:76: Requires package 'dnspython'
SKIPPED [1] tests/test_net.py:83: Requires package 'dnspython'
SKIPPED [1] tests/test_net.py:66: Requires package 'dnspython'
SKIPPED [1] tests/test_net.py:71: Requires package 'dnspython'
SKIPPED [1] /home/user/privex-helpers/tests/test_net.py:56: Skipping test TestGeneral.
 ↪test_ping_v6 as platform is
not supported: "privex.helpers.net.ping is not fully supported on platform 'Darwin'...
 ↪"
================== 94 passed, 5 skipped, 1 warnings in 21.66s ===================
```

3.19.3 Running individual test modules

Some test modules such as `test_cache` can be quite slow, as sometimes it's required to call sleep, e.g. `sleep(2)` either to prevent interference from previous/following tests, or when testing that an expiration/timeout works.

Thankfully, PyTest allows you to run individual test modules like this:

```
user@host: ~/privex-helpers $ pytest -rxXs -v tests/test_parse.py

============================= test session starts ==============================
platform darwin -- Python 3.7.0, pytest-5.2.2, py-1.8.0, pluggy-0.13.0
cachedir: .pytest_cache
rootdir: /home/user/privex-helpers
plugins: cov-2.8.1
collected 10 items

tests/test_parse.py::TestParseHelpers::test_csv_single PASSED            [ 10%]
tests/test_parse.py::TestParseHelpers::test_csv_spaced PASSED            [ 20%]
tests/test_parse.py::TestParseHelpers::test_env_bool_false PASSED        [ 30%]
tests/test_parse.py::TestParseHelpers::test_env_bool_true PASSED         [ 40%]
tests/test_parse.py::TestParseHelpers::test_env_nonexist_bool PASSED     [ 50%]
tests/test_parse.py::TestParseHelpers::test_kval_clean PASSED            [ 60%]
tests/test_parse.py::TestParseHelpers::test_kval_custom_clean PASSED     [ 70%]
tests/test_parse.py::TestParseHelpers::test_kval_custom_spaced PASSED    [ 80%]
tests/test_parse.py::TestParseHelpers::test_kval_single PASSED           [ 90%]
tests/test_parse.py::TestParseHelpers::test_kval_spaced PASSED           [100%]

============================== 10 passed in 0.09s ==============================
```

3.19.4 Running directly using Python Unittest

Alternatively, you can run the tests by hand with `python3.7` (or just `python3`), however we strongly recommend using PyTest as our tests use various PyTest functionality to allow for things such as skipping tests when you don't have a certain dependency installed.

Running via python unittest

```
user@the-matrix ~/privex-helpers $ python3.7 -m tests
.........................
----------------------------------------------------------------------
Ran 28 tests in 0.001s

OK
```

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

```
user@the-matrix ~/privex-helpers $ python3 -m tests -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)
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
```

Copyright:

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


    +=====================================================+
    |                © 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.
```

3.20 Unit Test List / Overview

tests.asyncx	
tests.base	Various classes / functions / attributes used by test cases (no actual test cases in here)
tests.cache	
tests.general	General test cases for various un-categorized functions / classes e.g.
tests.test_bool	Test cases for boolean helper functions, such as *is_true()*, *is_false()*, and *empty()*
tests.test_cache	Test cases for the cache decorator *r_cache()* plus cache layers `RedisCache` and `MemoryCache`
tests.test_collections	Test cases for *privex.helpers.collections*
tests.test_converters	
tests.test_crypto	Test cases for the *privex.helpers.crypto* module
tests.test_extras	Test cases for *privex.helpers.extras*
tests.test_parse	Test cases for parsing functions, such as *parse_csv()*, *env_keyval()* etc.
tests.test_rdns	A thorough test case for *ip_to_rdns()* - which converts IPv4/v6 addresses into ARPA reverse DNS domains.
tests.test_net	Test cases related to *privex.helpers.net* or generally network related functions such as *ping()*

3.20.1 tests.asyncx

3.20.2 tests.base

Various classes / functions / attributes used by test cases (no actual test cases in here)

Copyright:

Classes

EmptyIter()	A mock iterable object with zero length for testing empty()
PrivexBaseCase([methodName])	Base test-case for module test cases to inherit.

3.20.2.1 EmptyIter

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

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

3.20.2.1.1 Methods

Methods

—

3.20.2.2 PrivexBaseCase

class tests.base.**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.

3.20.2.2.1 Methods

Methods

—

3.20.2.2.2 Attributes

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
truthy	Truthful values, as various types

3.20.2.2.2.1 empty_lst

PrivexBaseCase.**empty_lst = [None, '', [], (), set(), {}, <tests.base.EmptyIter object>]**

3.20.2.2.2.2 empty_vals

PrivexBaseCase.**empty_vals = [None, '']**

3.20.2.2.2.3 empty_zero

PrivexBaseCase.**empty_zero = [None, '', 0, '0']**

3.20.2.2.2.4 falsey

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

3.20.2.2.2.5 falsey_empty

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

3.20.2.2.2.6 truthy

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

3.20.3 tests.cache

3.20.4 tests.general

General test cases for various un-categorized functions / classes e.g. *chunked()* and *inject_items()*

Copyright:

```
    +===================================================+
    |                 © 2019 Privex Inc.                |
    |               https://www.privex.io               |
    +===================================================+
    |                                                   |
    |        Originally Developed by Privex Inc.        |
    |        License: X11 / MIT                         |
    |                                                   |
    |        Core Developer(s):                         |
    |                                                   |
    |          (+)  Chris (@someguy123) [Privex]        |
    |          (+)  Kale (@kryogenic) [Privex]          |
    |                                                   |
    +===================================================+
```

(continues on next page)

(continued from previous page)

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

3.20.5 tests.test_bool

Test cases for boolean helper functions, such as *is_true()*, *is_false()*, and *empty()*

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.
```

Classes

TestBoolHelpers([methodName])	Test the boolean check functions is_true, is_false, as well as empty()

3.20.5.1 TestBoolHelpers

class tests.test_bool.**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.

3.20.5.1.1 Methods

Methods

test_empty_combined()	Test *empty()* with empty iterables AND different representations of 0
test_empty_lst()	Test *empty()* with empty iterables
test_empty_vals()	Test *empty()* with empty values
test_empty_zero()	Test *empty()* with different representations of 0
test_emptyif_only_empty()	
test_emptyif_only_value()	
test_emptyif_with_is_not_empty()	
test_isfalse_falsey()	Test *is_false()* with falsey values
test_isfalse_truthy()	Test *is_false()* with truthy values
test_istrue_falsey()	Test *is_true()* with falsey values
test_istrue_truthy()	Test *is_true()* with truthy values
test_notempty()	Test *empty()* with non-empty values

3.20.5.1.1.1 test_empty_combined

TestBoolHelpers.**test_empty_combined**()
Test *empty()* with empty iterables AND different representations of 0

3.20.5.1.1.2 test_empty_lst

TestBoolHelpers.**test_empty_lst**()
Test *empty()* with empty iterables

3.20.5.1.1.3 test_empty_vals

TestBoolHelpers.**test_empty_vals**()
Test *empty()* with empty values

3.20.5.1.1.4 test_empty_zero

TestBoolHelpers.**test_empty_zero**()
Test *empty()* with different representations of 0

3.20.5.1.1.5 test_emptyif_only_empty

TestBoolHelpers.**test_emptyif_only_empty**()

3.20.5.1.1.6 test_emptyif_only_value

TestBoolHelpers.**test_emptyif_only_value**()

3.20.5.1.1.7 test_emptyif_with_is_not_empty

TestBoolHelpers.**test_emptyif_with_is_not_empty**()

3.20.5.1.1.8 test_isfalse_falsey

TestBoolHelpers.**test_isfalse_falsey**()
Test *is_false()* with falsey values

3.20.5.1.1.9 test_isfalse_truthy

TestBoolHelpers.**test_isfalse_truthy**()
Test *is_false()* with truthy values

3.20.5.1.1.10 test_istrue_falsey

TestBoolHelpers.**test_istrue_falsey**()
Test *is_true()* with falsey values

3.20.5.1.1.11 test_istrue_truthy

TestBoolHelpers.**test_istrue_truthy**()
Test *is_true()* with truthy values

3.20.5.1.1.12 test_notempty

TestBoolHelpers.**test_notempty**()
Test *empty()* with non-empty values

3.20.5.1.2 Attributes

Attributes

—

3.20.6 tests.test_cache

Test cases for the cache decorator *r_cache()* plus cache layers RedisCache and MemoryCache

Copyright:

Classes

TestCacheDecoratorMemory([methodName])	Test that the decorator *privex.helpers.decorators.r_cache()* caches correctly, with adapter `privex.helpers.cache.MemoryCache.MemoryCache` and also verifies dynamic cache key generation works as expected.
TestCacheDecoratorRedis([methodName])	Test decorator *privex.helpers.decorators.r_cache()* with adapter `privex.helpers.cache.RedisCache.RedisCache`
TestMemoryCache([methodName])	`MemoryCache` Test cases for caching related functions/classes in *privex.helpers.cache*
TestRedisCache([methodName])	`RedisCache` Test cases for caching related functions/classes in *privex.helpers.cache*

3.20.6.1 TestCacheDecoratorMemory

class tests.test_cache.**TestCacheDecoratorMemory**(*methodName='runTest'*)

Test that the decorator *privex.helpers.decorators.r_cache()* caches correctly, with adapter `privex.helpers.cache.MemoryCache.MemoryCache` and also verifies dynamic cache key generation works as expected.

__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.

3.20.6.1.1 Methods

Methods

setUpClass()	Hook method for setting up class fixture before running tests in the class.
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

3.20.6.1.1.1 setUpClass

classmethod TestCacheDecoratorMemory.**setUpClass**()
Hook method for setting up class fixture before running tests in the class.

3.20.6.1.1.2 tearDown

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

3.20.6.1.1.3 test_rcache_callable

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

3.20.6.1.1.4 test_rcache_rand

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

3.20.6.1.1.5 test_rcache_rand_dynamic

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

3.20.6.1.2 Attributes

Attributes

cache

3.20.6.1.2.1 cache

TestCacheDecoratorMemory.**cache = <privex.helpers.cache.CacheWrapper object>**

3.20.6.2 TestCacheDecoratorRedis

class tests.test_cache.**TestCacheDecoratorRedis**(*methodName='runTest'*)
Test decorator *privex.helpers.decorators.r_cache()* with adapter privex.helpers.cache.RedisCache.RedisCache

(See *TestCacheDecoratorMemory*)

__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.

3.20.6.2.1 Methods

Methods

setUpClass()	Hook method for setting up class fixture before running tests in the class.

3.20.6.2.1.1 setUpClass

classmethod TestCacheDecoratorRedis.**setUpClass**()
Hook method for setting up class fixture before running tests in the class.

3.20.6.2.2 Attributes

Attributes

pytestmark

3.20.6.2.2.1 pytestmark

TestCacheDecoratorRedis.**pytestmark = [Mark(name='skipif', args=(False,), kwargs={'reason':**

3.20.6.3 TestMemoryCache

class tests.test_cache.**TestMemoryCache**(*methodName='runTest'*)
MemoryCache Test cases for caching related functions/classes in *privex.helpers.cache*

__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.

3.20.6.3.1 Methods

Methods

setUpClass()	Set the current cache adapter to an instance of MemoryCache() and make it available through `self.cache`
tearDownClass()	Hook method for deconstructing the class fixture after running all tests in the class.
test_cache_expire()	Test that cache keys are removed after the specified timeout
test_cache_remove()	Test that cache.remove correctly removes cache keys
test_cache_set()	Test basic cache.set and cache.get
test_cache_update_timeout()	Test that cache.update_timeout extends timeouts correctly

continues on next page

Table 51 – continued from previous page

test_cache_update_timeout_raise()	Test that cache.update_timeout raises `CacheNotFound` if the key does not exist

3.20.6.3.1.1 setUpClass

classmethod `TestMemoryCache.`**`setUpClass`**`()`
Set the current cache adapter to an instance of MemoryCache() and make it available through `self.cache`

3.20.6.3.1.2 tearDownClass

classmethod `TestMemoryCache.`**`tearDownClass`**`()`
Hook method for deconstructing the class fixture after running all tests in the class.

3.20.6.3.1.3 test_cache_expire

`TestMemoryCache.`**`test_cache_expire`**`()`
Test that cache keys are removed after the specified timeout

3.20.6.3.1.4 test_cache_remove

`TestMemoryCache.`**`test_cache_remove`**`()`
Test that cache.remove correctly removes cache keys

3.20.6.3.1.5 test_cache_set

`TestMemoryCache.`**`test_cache_set`**`()`
Test basic cache.set and cache.get

3.20.6.3.1.6 test_cache_update_timeout

`TestMemoryCache.`**`test_cache_update_timeout`**`()`
Test that cache.update_timeout extends timeouts correctly

3.20.6.3.1.7 test_cache_update_timeout_raise

`TestMemoryCache.`**`test_cache_update_timeout_raise`**`()`
Test that cache.update_timeout raises `CacheNotFound` if the key does not exist

3.20.6.3.2 Attributes

Attributes

cache_keys	A list of all cache keys used during the test case, so they can be removed by *tearDown()* once done.

3.20.6.3.2.1 cache_keys

TestMemoryCache.**cache_keys = ['test_cache_set', 'test_expire', 'test_update_timeout', 'test**
A list of all cache keys used during the test case, so they can be removed by *tearDown()* once done.

3.20.6.4 TestRedisCache

class tests.test_cache.**TestRedisCache**(*methodName='runTest'*)
RedisCache Test cases for caching related functions/classes in *privex.helpers.cache*

This is **simply a child class** for *TestMemoryCache* - but with an overridden `setUpClass` to ensure the cache adapter is set to `RedisCache` for this re-run.

__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.

3.20.6.4.1 Methods

Methods

setUpClass()	Set the current cache adapter to an instance of RedisCache() and make it available through `self.cache`

3.20.6.4.1.1 setUpClass

classmethod TestRedisCache.**setUpClass**()
Set the current cache adapter to an instance of RedisCache() and make it available through `self.cache`

3.20.6.4.2 Attributes

Attributes

pytestmark

3.20.6.4.2.1 pytestmark

```
TestRedisCache.pytestmark = [Mark(name='skipif', args=(False,), kwargs={'reason':  "TestRe
```

3.20.7 tests.test_collections

Test cases for *privex.helpers.collections*

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.
```

Classes

TestDictObject([methodName])	
TestDictableNamedtuple([methodName])	Test the function *dictable_namedtuple()* and compare it against collections.namedtuple() to ensure that dictable_namedtuple's should be backwards compatible with code that takes namedtuple's.

continues on next page

Table 55 – continued from previous page

TestIsNamedTuple([methodName])	Test the function *is_namedtuple()* against various different types and objects to ensure it returns `True` or `False` appropriately, with tests containing a mixture of both `collections.namedtuple()` and *dictable_namedtuple()*.
TestOrderedDictObject([methodName])	

3.20.7.1 TestDictObject

class tests.test_collections.**TestDictObject**(*methodName='runTest'*)

__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.

3.20.7.1.1 Methods

Methods

test_convert_from_dict()	Test converting a `dict` into a *DictObject*
test_convert_to_dict()	Test converting a *DictObject* into a `dict`
test_json_dumps()	Test serializing a simple *DictObject* into JSON
test_json_dumps_nested()	Test serializing a *DictObject* with a nested *DictObject* into JSON
test_set_attr()	Test setting a dictionary key via an attribute `x.y = 123`
test_set_item()	Test setting a dictionary key via an item/key `x['y'] = 123`

3.20.7.1.1.1 test_convert_from_dict

TestDictObject.**test_convert_from_dict**()
Test converting a `dict` into a *DictObject*

3.20.7.1.1.2 test_convert_to_dict

TestDictObject.**test_convert_to_dict**()
Test converting a *DictObject* into a dict

3.20.7.1.1.3 test_json_dumps

TestDictObject.**test_json_dumps**()
Test serializing a simple *DictObject* into JSON

3.20.7.1.1.4 test_json_dumps_nested

TestDictObject.**test_json_dumps_nested**()
Test serializing a *DictObject* with a nested *DictObject* into JSON

3.20.7.1.1.5 test_set_attr

TestDictObject.**test_set_attr**()
Test setting a dictionary key via an attribute `x.y = 123`

3.20.7.1.1.6 test_set_item

TestDictObject.**test_set_item**()
Test setting a dictionary key via an item/key `x['y'] = 123`

3.20.7.1.2 Attributes

Attributes

—

3.20.7.2 TestDictableNamedtuple

class tests.test_collections.**TestDictableNamedtuple**(*methodName='runTest'*)
Test the function *dictable_namedtuple()* and compare it against `collections.namedtuple()` to ensure that dictable_namedtuple's should be backwards compatible with code that takes namedtuple's.

Also tests new functionality that only exists in dictable_namedtuple's, and compares it against standard namedtuples, including:

- Test getting by item/key, i.e. `john['first_name']`, and confirm normal namedtuples raise exceptions
- Test setting new item/key's and attributes `item['color'] = 'Brown'`, and confirm normal namedtuples raise exceptions
- Test casting dictable namedtuple's to dict's `dict(item)`, and confirm normal namedtuples raise exceptions

__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.

3.20.7.2.1 Methods

Methods

setUp()	At the start of each test, reset *example_items*, Item and NmItem in-case any tests have modified them
test_asdict()	Test ._asdict works on both dictable + normal namedtuple
test_convert()	Test convert_dictable_namedtuple converts example namedtuple instance into a dictable_namedtuple instance
test_dict_cast()	Test casting dictable_namedtuple using dict works as expected, but fails on normal namedtuple
test_get_attr()	Confirm getting attributes is equivalent on dictable namedtuple to standard namedtuple
test_get_index()	Test we can access by integer index on dictable + normal namedtuple
test_get_item()	Test we can access named items on on dictable namedtuple while standard namedtuple raises exceptions
test_metadata()	Confirm sameness of class/instance metadata such as class name/qualname, module name, and stringification between *dictable_namedtuple()* and collections.namedtuple()
test_set_attr()	Test that we can create the new attribute color on the dictable_namedtuple
test_set_item()	Test that we can create the new item/key color on the dictable_namedtuple
test_subclass()	Test subclass_dictable_namedtuple converts NmItem into a dictable_namedtuple type

3.20.7.2.1.1 setUp

TestDictableNamedtuple.**setUp**()

At the start of each test, reset *example_items*, Item and NmItem in-case any tests have modified them

3.20.7.2.1.2 test_asdict

TestDictableNamedtuple.**test_asdict**()
Test `._asdict` works on both dictable + normal namedtuple

3.20.7.2.1.3 test_convert

TestDictableNamedtuple.**test_convert**()
Test convert_dictable_namedtuple converts example namedtuple instance into a dictable_namedtuple instance

3.20.7.2.1.4 test_dict_cast

TestDictableNamedtuple.**test_dict_cast**()
Test casting dictable_namedtuple using `dict` works as expected, but fails on normal namedtuple

3.20.7.2.1.5 test_get_attr

TestDictableNamedtuple.**test_get_attr**()
Confirm getting attributes is equivalent on dictable namedtuple to standard namedtuple

3.20.7.2.1.6 test_get_index

TestDictableNamedtuple.**test_get_index**()
Test we can access by integer index on dictable + normal namedtuple

3.20.7.2.1.7 test_get_item

TestDictableNamedtuple.**test_get_item**()
Test we can access named items on on dictable namedtuple while standard namedtuple raises exceptions

3.20.7.2.1.8 test_metadata

TestDictableNamedtuple.**test_metadata**()
Confirm sameness of class/instance metadata such as class name/qualname, module name, and stringification between *`dictable_namedtuple()`* and `collections.namedtuple()`

3.20.7.2.1.9 test_set_attr

TestDictableNamedtuple.**test_set_attr**()
Test that we can create the new attribute `color` on the dictable_namedtuple

3.20.7.2.1.10 test_set_item

TestDictableNamedtuple.**test_set_item**()
Test that we can create the new item/key `color` on the dictable_namedtuple

3.20.7.2.1.11 test_subclass

TestDictableNamedtuple.**test_subclass**()
Test subclass_dictable_namedtuple converts `NmItem` into a dictable_namedtuple type

3.20.7.2.2 Attributes

Attributes

example_items	A tuple containing an instance of both `Item` and `NmItem`

3.20.7.2.2.1 example_items

TestDictableNamedtuple.**example_items = (Item(name='Box', description='Small Cardboard Box'**
A tuple containing an instance of both `Item` and `NmItem`

3.20.7.3 TestIsNamedTuple

class tests.test_collections.**TestIsNamedTuple**(*methodName='runTest'*)
Test the function *is_namedtuple()* against various different types and objects to ensure it returns `True` or `False` appropriately, with tests containing a mixture of both `collections.namedtuple()` and *dictable_namedtuple()*.

__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.

3.20.7.3.1 Methods

Methods

test_dictable_namedtuple()	Test *is_namedtuple()* returns `True` when all arguments are valid dictable_namedtuple's
test_dictable_namedtuple_plus_invalid()	(dictable_namedtuples) Test *is_namedtuple()* returns `False` when some arguments are NOT namedtuple's
test_dictable_plus_normal_namedtuple()	Test *is_namedtuple()* returns `True` when arguments are a mix of namedtuple + dictable_namedtuple's
test_not_namedtuple_class()	Test that classes and instances of classes are not namedtuple's

continues on next page

Table 60 – continued from previous page

test_not_namedtuple_dict()	Test that dictionaries are not namedtuple's
test_not_namedtuple_float()	Test that float's are not namedtuple's
test_not_namedtuple_int()	Test that int's are not namedtuple's
test_not_namedtuple_list()	Test that list's are not namedtuple's
test_not_namedtuple_tuple()	Test that tuple's are not namedtuple's
test_real_namedtuple()	Test *is_namedtuple()* returns `True` when all arguments are valid namedtuple's
test_real_namedtuple_plus_invalid()	(namedtuples) Test *is_namedtuple()* returns `False` when some arguments are NOT namedtuple's

3.20.7.3.1.1 test_dictable_namedtuple

`TestIsNamedTuple.`**`test_dictable_namedtuple`**`()`
Test *is_namedtuple()* returns `True` when all arguments are valid dictable_namedtuple's

3.20.7.3.1.2 test_dictable_namedtuple_plus_invalid

`TestIsNamedTuple.`**`test_dictable_namedtuple_plus_invalid`**`()`
(dictable_namedtuples) Test *is_namedtuple()* returns `False` when some arguments are NOT namedtuple's

3.20.7.3.1.3 test_dictable_plus_normal_namedtuple

`TestIsNamedTuple.`**`test_dictable_plus_normal_namedtuple`**`()`
Test *is_namedtuple()* returns `True` when arguments are a mix of namedtuple + dictable_namedtuple's

3.20.7.3.1.4 test_not_namedtuple_class

`TestIsNamedTuple.`**`test_not_namedtuple_class`**`()`
Test that classes and instances of classes are not namedtuple's

3.20.7.3.1.5 test_not_namedtuple_dict

`TestIsNamedTuple.`**`test_not_namedtuple_dict`**`()`
Test that dictionaries are not namedtuple's

3.20.7.3.1.6 test_not_namedtuple_float

`TestIsNamedTuple.`**`test_not_namedtuple_float`**`()`
Test that float's are not namedtuple's

3.20.7.3.1.7 test_not_namedtuple_int

TestIsNamedTuple.**test_not_namedtuple_int**()
Test that int's are not namedtuple's

3.20.7.3.1.8 test_not_namedtuple_list

TestIsNamedTuple.**test_not_namedtuple_list**()
Test that list's are not namedtuple's

3.20.7.3.1.9 test_not_namedtuple_tuple

TestIsNamedTuple.**test_not_namedtuple_tuple**()
Test that tuple's are not namedtuple's

3.20.7.3.1.10 test_real_namedtuple

TestIsNamedTuple.**test_real_namedtuple**()
Test *is_namedtuple()* returns `True` when all arguments are valid namedtuple's

3.20.7.3.1.11 test_real_namedtuple_plus_invalid

TestIsNamedTuple.**test_real_namedtuple_plus_invalid**()
(namedtuples) Test *is_namedtuple()* returns `False` when some arguments are NOT namedtuple's

3.20.7.3.2 Attributes

Attributes

dict_persons
named_persons

3.20.7.3.2.1 dict_persons

TestIsNamedTuple.**dict_persons = (Person(first_name='John', last_name='Doe'), Person(first_**

3.20.7.3.2.2 named_persons

`TestIsNamedTuple.named_persons = (Person(first_name='John', last_name='Doe'), Person(first_`

3.20.7.4 TestOrderedDictObject

class tests.test_collections.**TestOrderedDictObject**(*methodName='runTest'*)

__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.

3.20.7.4.1 Methods

Methods

test_convert_from_dict()	Test converting a `dict` into a *DictObject*
test_convert_to_dict()	Test converting a *OrderedDictObject* into a `dict`
test_json_dumps()	Test serializing a simple *OrderedDictObject* into JSON
test_json_dumps_nested()	Test serializing a *OrderedDictObject* with a nested *OrderedDictObject* into JSON
test_set_attr()	Test setting a dictionary key via an attribute `x.y = 123`
test_set_item()	Test setting a dictionary key via an item/key `x['y'] = 123`

3.20.7.4.1.1 test_convert_from_dict

TestOrderedDictObject.**test_convert_from_dict**()
Test converting a `dict` into a *DictObject*

3.20.7.4.1.2 test_convert_to_dict

TestOrderedDictObject.**test_convert_to_dict**()
Test converting a *OrderedDictObject* into a `dict`

3.20.7.4.1.3 test_json_dumps

TestOrderedDictObject.**test_json_dumps**()
Test serializing a simple *OrderedDictObject* into JSON

3.20.7.4.1.4 test_json_dumps_nested

TestOrderedDictObject.**test_json_dumps_nested**()
Test serializing a *OrderedDictObject* with a nested *OrderedDictObject* into JSON

3.20.7.4.1.5 test_set_attr

TestOrderedDictObject.**test_set_attr**()
Test setting a dictionary key via an attribute `x.y = 123`

3.20.7.4.1.6 test_set_item

TestOrderedDictObject.**test_set_item**()
Test setting a dictionary key via an item/key `x['y'] = 123`

3.20.7.4.2 Attributes

Attributes

—

3.20.8 tests.test_converters

Classes

TestConvertDate([methodName])	Test cases for date/time converter functions/classes
TestConvertGeneral([methodName])	Test cases for general converter functions/classes

3.20.8.1 TestConvertDate

class tests.test_converters.**TestConvertDate**(*methodName='runTest'*)
Test cases for date/time converter functions/classes

__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.

3.20.8.1.1 Methods

Methods

test_convert_date_int()	Test *convert_datetime()* - converting integer unix time into datetime
test_convert_date_int_ms()	Test *convert_datetime()* - converting integer unix time (milliseconds) into datetime

continues on next page

Table 65 – continued from previous page

test_convert_date_int_str()	Test *convert_datetime()* - converting string unix time into datetime
test_convert_date_str()	Test converting ISO date string into datetime
test_convert_date_str_2()	Test converting no-timezone ISO date string into datetime
test_convert_date_str_3()	Test converting no-timezone human date string (spaces) into datetime
test_convert_date_str_4()	Test converting no-timezone human date string (slashes) into datetime
test_convert_unixtime_int()	Test *convert_unixtime_datetime()* - converting integer unix time into datetime
test_convert_unixtime_int_ms()	Test *convert_datetime()* - converting integer unix time (milliseconds) into datetime
test_convert_unixtime_int_str()	Test *convert_unixtime_datetime()* - converting string unix time into datetime

3.20.8.1.1.1 test_convert_date_int

TestConvertDate.**test_convert_date_int**()
Test *convert_datetime()* - converting integer unix time into datetime

3.20.8.1.1.2 test_convert_date_int_ms

TestConvertDate.**test_convert_date_int_ms**()
Test *convert_datetime()* - converting integer unix time (milliseconds) into datetime

3.20.8.1.1.3 test_convert_date_int_str

TestConvertDate.**test_convert_date_int_str**()
Test *convert_datetime()* - converting string unix time into datetime

3.20.8.1.1.4 test_convert_date_str

TestConvertDate.**test_convert_date_str**()
Test converting ISO date string into datetime

3.20.8.1.1.5 test_convert_date_str_2

TestConvertDate.**test_convert_date_str_2**()
Test converting no-timezone ISO date string into datetime

3.20.8.1.1.6 test_convert_date_str_3

TestConvertDate.**test_convert_date_str_3**()
Test converting no-timezone human date string (spaces) into datetime

3.20.8.1.1.7 test_convert_date_str_4

TestConvertDate.**test_convert_date_str_4**()
Test converting no-timezone human date string (slashes) into datetime

3.20.8.1.1.8 test_convert_unixtime_int

TestConvertDate.**test_convert_unixtime_int**()
Test *convert_unixtime_datetime()* - converting integer unix time into datetime

3.20.8.1.1.9 test_convert_unixtime_int_ms

TestConvertDate.**test_convert_unixtime_int_ms**()
Test *convert_datetime()* - converting integer unix time (milliseconds) into datetime

3.20.8.1.1.10 test_convert_unixtime_int_str

TestConvertDate.**test_convert_unixtime_int_str**()
Test *convert_unixtime_datetime()* - converting string unix time into datetime

3.20.8.1.2 Attributes

Attributes

—

3.20.8.2 TestConvertGeneral

class tests.test_converters.**TestConvertGeneral**(*methodName='runTest'*)
Test cases for general converter functions/classes

__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.

3.20.8.2.1 Methods

Methods

test_convert_bool_int_empty()
test_convert_bool_int_empty_cust()
test_convert_bool_int_empty_fail()
test_convert_bool_int_false()
test_convert_bool_int_true()
test_convert_int_bool_empty()
test_convert_int_bool_empty_cust()
test_convert_int_bool_empty_fail()
test_convert_int_bool_false()
test_convert_int_bool_true()

3.20.8.2.1.1 test_convert_bool_int_empty

TestConvertGeneral.**test_convert_bool_int_empty**()

3.20.8.2.1.2 test_convert_bool_int_empty_cust

TestConvertGeneral.**test_convert_bool_int_empty_cust**()

3.20.8.2.1.3 test_convert_bool_int_empty_fail

TestConvertGeneral.**test_convert_bool_int_empty_fail**()

3.20.8.2.1.4 test_convert_bool_int_false

TestConvertGeneral.**test_convert_bool_int_false**()

3.20.8.2.1.5 test_convert_bool_int_true

TestConvertGeneral.**test_convert_bool_int_true**()

3.20.8.2.1.6 test_convert_int_bool_empty

TestConvertGeneral.**test_convert_int_bool_empty**()

3.20.8.2.1.7 test_convert_int_bool_empty_cust

TestConvertGeneral.**test_convert_int_bool_empty_cust**()

3.20.8.2.1.8 test_convert_int_bool_empty_fail

TestConvertGeneral.**test_convert_int_bool_empty_fail**()

3.20.8.2.1.9 test_convert_int_bool_false

TestConvertGeneral.**test_convert_int_bool_false**()

3.20.8.2.1.10 test_convert_int_bool_true

TestConvertGeneral.**test_convert_int_bool_true**()

3.20.8.2.2 Attributes

Attributes

3.20.9 tests.test_crypto

Test cases for the *privex.helpers.crypto* module

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:
```

(continues on next page)

(continued from previous page)

```
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.
```

Classes

CryptoBaseCase([methodName])	
TestEncryptHelper([methodName])	Test `EncryptHelper` key generation, encryption, decryption and more
TestKeyManagerGeneration([methodName])	
TestKeyManagerLoad([methodName])	Test `KeyManager` asymmetric key loading
TestKeyManagerSignVerifyEncrypt([methodName])	Test `KeyManager` asymmetric key signing/verification, and encryption/decryption

3.20.9.1 CryptoBaseCase

class tests.test_crypto.**CryptoBaseCase**(*methodName='runTest'*)

__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.

3.20.9.1.1 Methods

Methods

—

3.20.9.1.2 Attributes

Attributes

fake_b64_key

3.20.9.1.2.1 fake_b64_key

CryptoBaseCase.**fake_b64_key = 'bm90IGEga2V5'**

3.20.9.2 TestEncryptHelper

class tests.test_crypto.**TestEncryptHelper**(*methodName='runTest'*)

Test EncryptHelper key generation, encryption, decryption and more

__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.

3.20.9.2.1 Methods

Methods

test_corrupt_key_encrypt()	Test that encrypt_str fails when using a corrupted key
test_generate_key_enc_dec()	Test EncryptHelper.generate_key() key works for encryption and decryption
test_invalid_key_decrypt()	Test that decrypt_str fails when using the wrong key
test_is_encrypted()	Test that is_encrypted returns True for encrypted data, and False for non-encrypted
test_password_key_diffpass()	Test that password_key returns two different keys for two passwords with the same salt
test_password_key_diffsalt()	Test that password_key returns two different keys for passwords with different salts
test_password_key_equal()	Test that password_key returns the same key when ran with the same arguments
test_password_key_gensalt()	Test that we can reproduce the same key from password_key's auto-generated salt

3.20.9.2.1.1 test_corrupt_key_encrypt

TestEncryptHelper.**test_corrupt_key_encrypt**()

Test that encrypt_str fails when using a corrupted key

3.20.9.2.1.2 test_generate_key_enc_dec

TestEncryptHelper.**test_generate_key_enc_dec**()
Test `EncryptHelper.generate_key()` key works for encryption and decryption

3.20.9.2.1.3 test_invalid_key_decrypt

TestEncryptHelper.**test_invalid_key_decrypt**()
Test that decrypt_str fails when using the wrong key

3.20.9.2.1.4 test_is_encrypted

TestEncryptHelper.**test_is_encrypted**()
Test that is_encrypted returns True for encrypted data, and False for non-encrypted

3.20.9.2.1.5 test_password_key_diffpass

TestEncryptHelper.**test_password_key_diffpass**()
Test that password_key returns two different keys for two passwords with the same salt

3.20.9.2.1.6 test_password_key_diffsalt

TestEncryptHelper.**test_password_key_diffsalt**()
Test that password_key returns two different keys for passwords with different salts

3.20.9.2.1.7 test_password_key_equal

TestEncryptHelper.**test_password_key_equal**()
Test that password_key returns the same key when ran with the same arguments

3.20.9.2.1.8 test_password_key_gensalt

TestEncryptHelper.**test_password_key_gensalt**()
Test that we can reproduce the same key from password_key's auto-generated salt

3.20.9.2.2 Attributes

Attributes

txt

3.20.9.2.2.1 txt

TestEncryptHelper.**txt = 'This is a test.'**

3.20.9.3 TestKeyManagerGeneration

class tests.test_crypto.**TestKeyManagerGeneration**(*methodName='runTest'*)

__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.

3.20.9.3.1 Methods

Methods

test_ecdsa_gen()	Generate an ECDSA keypair, check the pub/priv lengths, and confirm they're formatted correctly
test_ed25519_gen()	Generate an Ed25519 keypair, check the pub/priv lengths, and confirm they're formatted correctly
test_output_keypair()	Test outputting a keypair to files creates files, and file contents match the returned priv/pub
test_rsa_gen()	Generate an RSA 2048 + 4096-bit key, check the pub/priv lengths, and confirm they're formatted correctly

3.20.9.3.1.1 test_ecdsa_gen

TestKeyManagerGeneration.**test_ecdsa_gen**()
Generate an ECDSA keypair, check the pub/priv lengths, and confirm they're formatted correctly

3.20.9.3.1.2 test_ed25519_gen

TestKeyManagerGeneration.**test_ed25519_gen**()
Generate an Ed25519 keypair, check the pub/priv lengths, and confirm they're formatted correctly

3.20.9.3.1.3 test_output_keypair

TestKeyManagerGeneration.**test_output_keypair**()
Test outputting a keypair to files creates files, and file contents match the returned priv/pub

3.20.9.3.1.4 test_rsa_gen

`TestKeyManagerGeneration.`**`test_rsa_gen`**`()`
Generate an RSA 2048 + 4096-bit key, check the pub/priv lengths, and confirm they're formatted correctly

3.20.9.3.2 Attributes

Attributes

—

3.20.9.4 TestKeyManagerLoad

`class` `tests.test_crypto.`**`TestKeyManagerLoad`**(*methodName='runTest'*)
Test `KeyManager` asymmetric key loading

`__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.

3.20.9.4.1 Methods

Methods

test_ecdsa_load()	Generate and attempt to load an ECDSA keypair
test_ed25519_load()	Generate and attempt to load an Ed25519 keypair
test_load_invalid()	Initialise KeyManager with an invalid key to confirm it raises InvalidFormat
test_load_keyfile_corrupt_private()	Test `KeyManager.load_keyfile()` raises `InvalidFormat` with corrupted PEM private key
test_load_keyfile_corrupt_public()	Test `KeyManager.load_keyfile()` raises `InvalidFormat` with corrupted public key
test_load_keyfile_corrupt_public_2()	Test `KeyManager.load_keyfile()` raises `InvalidFormat` with corrupted public key (but with valid b64)
test_load_keyfile_noexist()	Test `KeyManager.load_keyfile()` raises `FileNotFoundError` with non-existent path
test_load_keyfile_sign_verify_rsa()	Generate a key pair + save to disk, then load the keypair from disk.
test_rsa_load()	Generate and attempt to load an RSA keypair

3.20.9.4.1.1 test_ecdsa_load

TestKeyManagerLoad.**test_ecdsa_load**()
Generate and attempt to load an ECDSA keypair

3.20.9.4.1.2 test_ed25519_load

TestKeyManagerLoad.**test_ed25519_load**()
Generate and attempt to load an Ed25519 keypair

3.20.9.4.1.3 test_load_invalid

TestKeyManagerLoad.**test_load_invalid**()
Initialise KeyManager with an invalid key to confirm it raises InvalidFormat

3.20.9.4.1.4 test_load_keyfile_corrupt_private

TestKeyManagerLoad.**test_load_keyfile_corrupt_private**()
Test `KeyManager.load_keyfile()` raises `InvalidFormat` with corrupted PEM private key

3.20.9.4.1.5 test_load_keyfile_corrupt_public

TestKeyManagerLoad.**test_load_keyfile_corrupt_public**()
Test `KeyManager.load_keyfile()` raises `InvalidFormat` with corrupted public key

3.20.9.4.1.6 test_load_keyfile_corrupt_public_2

TestKeyManagerLoad.**test_load_keyfile_corrupt_public_2**()
Test `KeyManager.load_keyfile()` raises `InvalidFormat` with corrupted public key (but with valid b64)

3.20.9.4.1.7 test_load_keyfile_noexist

TestKeyManagerLoad.**test_load_keyfile_noexist**()
Test `KeyManager.load_keyfile()` raises `FileNotFoundError` with non-existent path

3.20.9.4.1.8 test_load_keyfile_sign_verify_rsa

TestKeyManagerLoad.**test_load_keyfile_sign_verify_rsa**()
Generate a key pair + save to disk, then load the keypair from disk. Confirm that the keys on disk definitely match the returned tuple by running signature verification.

Uses KeyManager with both the public/private keys from disk, and the output_keypair returned public/private keys

3.20.9.4.1.9 test_rsa_load

TestKeyManagerLoad.**test_rsa_load**()
Generate and attempt to load an RSA keypair

3.20.9.4.2 Attributes

Attributes

—

3.20.9.5 TestKeyManagerSignVerifyEncrypt

class tests.test_crypto.**TestKeyManagerSignVerifyEncrypt**(*methodName='runTest'*)
Test KeyManager asymmetric key signing/verification, and encryption/decryption

__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.

3.20.9.5.1 Methods

Methods

test_ecdsa_sign_verify()	Attempt to sign and verify a message using an ECDSA keypair using _sign_verify() test helper
test_ed25519_sign_verify()	Attempt to sign and verify a message using an Ed25519 keypair using _sign_verify() test helper
test_rsa_encrypt_decrypt()	
test_rsa_sign_verify()	Attempt to sign and verify a message using an RSA keypair using _sign_verify() test helper

3.20.9.5.1.1 test_ecdsa_sign_verify

TestKeyManagerSignVerifyEncrypt.**test_ecdsa_sign_verify**()
Attempt to sign and verify a message using an ECDSA keypair using _sign_verify() test helper

3.20.9.5.1.2 test_ed25519_sign_verify

TestKeyManagerSignVerifyEncrypt.**test_ed25519_sign_verify**()
Attempt to sign and verify a message using an Ed25519 keypair using `_sign_verify()` test helper

3.20.9.5.1.3 test_rsa_encrypt_decrypt

TestKeyManagerSignVerifyEncrypt.**test_rsa_encrypt_decrypt**()

3.20.9.5.1.4 test_rsa_sign_verify

TestKeyManagerSignVerifyEncrypt.**test_rsa_sign_verify**()
Attempt to sign and verify a message using an RSA keypair using `_sign_verify()` test helper

3.20.9.5.2 Attributes

Attributes

—

3.20.10 tests.test_extras

Test cases for *privex.helpers.extras*

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.
```

(continues on next page)

(continued from previous page)

```
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.
```

Classes

Example(hello, testing)	
TestAttrs([methodName])	
TestGit([methodName])	

3.20.10.1 Example

class tests.test_extras.**Example**(*hello: str*, *testing: bool = True*)

__init__(*hello: str*, *testing: bool = True*) → None
Initialize self. See help(type(self)) for accurate signature.

3.20.10.1.1 Methods

Methods

__init__(hello[, testing])	Initialize self.

3.20.10.1.1.1 __init__

Example.**__init__**(*hello: str*, *testing: bool = True*) → None
Initialize self. See help(type(self)) for accurate signature.

3.20.10.2 TestAttrs

class tests.test_extras.**TestAttrs**(*methodName='runTest'*)

__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.

3.20.10.2.1 Methods

Methods

test_dictable_cast_dict()	Test casting an AttribDictable attrs instance to a dict
test_dictable_set_get()	Test setting and getting attributes on a AttribDictable attrs instance

3.20.10.2.1.1 test_dictable_cast_dict

TestAttrs.**test_dictable_cast_dict**()
Test casting an AttribDictable attrs instance to a dict

3.20.10.2.1.2 test_dictable_set_get

TestAttrs.**test_dictable_set_get**()
Test setting and getting attributes on a AttribDictable attrs instance

3.20.10.2.2 Attributes

Attributes

pytestmark

3.20.10.2.2.1 pytestmark

TestAttrs.**pytestmark = [Mark(name='skipif', args=(False,), kwargs={'reason': 'extras.HAS_**

3.20.10.3 TestGit

class tests.test_extras.**TestGit**(*methodName='runTest'*)

__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.

3.20.10.3.1 Methods

Methods

setUp()	Hook method for setting up the test fixture before exercising it.
tearDown()	Hook method for deconstructing the test fixture after testing it.
test_add()	
test_add_async()	
test_checkout()	
test_checkout_async()	
test_commit()	
test_commit_async()	
test_get_current_branch()	
test_get_current_commit()	
test_get_current_tag()	
test_init()	
test_init_async()	

3.20.10.3.1.1 setUp

TestGit.**setUp**() → None
Hook method for setting up the test fixture before exercising it.

3.20.10.3.1.2 tearDown

TestGit.**tearDown**() → None
Hook method for deconstructing the test fixture after testing it.

3.20.10.3.1.3 test_add

TestGit.**test_add**()

3.20.10.3.1.4 test_add_async

TestGit.**test_add_async**()

3.20.10.3.1.5 test_checkout

TestGit.**test_checkout**()

3.20.10.3.1.6 test_checkout_async

TestGit.**test_checkout_async**()

3.20.10.3.1.7 test_commit

TestGit.**test_commit**()

3.20.10.3.1.8 test_commit_async

TestGit.**test_commit_async**()

3.20.10.3.1.9 test_get_current_branch

TestGit.**test_get_current_branch**()

3.20.10.3.1.10 test_get_current_commit

TestGit.**test_get_current_commit**()

3.20.10.3.1.11 test_get_current_tag

TestGit.**test_get_current_tag**()

3.20.10.3.1.12 test_init

TestGit.**test_init**()

3.20.10.3.1.13 test_init_async

TestGit.**test_init_async**()

3.20.10.3.2 Attributes

Attributes

3.20.11 tests.test_parse

Test cases for parsing functions, such as *parse_csv()*, *env_keyval()* etc.

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.
```

Classes

TestParseHelpers([methodName])	Test the parsing functions parse_csv and parse_keyval

3.20.11.1 TestParseHelpers

class tests.test_parse.**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.

3.20.11.1.1 Methods

Methods

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

3.20.11.1.1.1 test_csv_single

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

3.20.11.1.1.2 test_csv_spaced

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

3.20.11.1.1.3 test_env_bool_false

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

3.20.11.1.1.4 test_env_bool_true

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

3.20.11.1.1.5 test_env_nonexist_bool

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

3.20.11.1.1.6 test_kval_clean

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

3.20.11.1.1.7 test_kval_custom_clean

TestParseHelpers.**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)

3.20.11.1.1.8 test_kval_custom_spaced

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

3.20.11.1.1.9 test_kval_single

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

3.20.11.1.1.10 test_kval_spaced

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

3.20.11.1.2 Attributes

Attributes

—

3.20.12 tests.test_rdns

A thorough test case for *ip_to_rdns()* - which converts IPv4/v6 addresses into ARPA reverse DNS domains.

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.
```

Classes

TestIPReverseDNS([methodName])	Unit testing for the reverse DNS functions in *privex.helpers.net*

3.20.12.1 TestIPReverseDNS

class tests.test_rdns.**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.

3.20.12.1.1 Methods

Methods

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

3.20.12.1.1.1 test_v4_arpa_boundary_16bit

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

3.20.12.1.1.2 test_v4_arpa_boundary_24bit

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

3.20.12.1.1.3 test_v4_inv_boundary

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

3.20.12.1.1.4 test_v4_inv_boundary_2

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

3.20.12.1.1.5 test_v4_invalid

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

3.20.12.1.1.6 test_v4_invalid_2

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

3.20.12.1.1.7 test_v4_to_arpa

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

3.20.12.1.1.8 test_v6_arpa_boundary_16bit

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

3.20.12.1.1.9 test_v6_arpa_boundary_32bit

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

3.20.12.1.1.10 test_v6_inv_boundary

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

3.20.12.1.1.11 test_v6_inv_boundary_2

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

3.20.12.1.1.12 test_v6_invalid

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

3.20.12.1.1.13 test_v6_invalid_2

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

3.20.12.1.1.14 test_v6_to_arpa

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

3.20.12.1.2 Attributes

Attributes

3.20.13 tests.test_net

Test cases related to *privex.helpers.net* or generally network related functions such as *ping()*

Classes

TestNet([methodName])	Test cases related to *privex.helpers.net* or generally network related functions
TestNetResolveIP([methodName])	Test cases related to *resolve_ips()*, *resolve_ip()* and *resolve_ips_multi()*

3.20.13.1 TestNet

class tests.test_net.**TestNet**(*methodName='runTest'*)

Test cases related to *privex.helpers.net* or generally network related functions

__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.

3.20.13.1.1 Methods

Methods

test_asn_to_name_erroneous()	Test asn_to_name returns 'Unknown ASN' when quiet, otherwise throws a KeyError for ASN 'nonexistent'
test_asn_to_name_erroneous_2()	Test asn_to_name returns 'Unknown ASN' when quiet, otherwise throws KeyError for the ASN 999999999
test_asn_to_name_int()	Test Privex's ASN (as an int) 210083 resolves to 'PRIVEX, SE'
test_asn_to_name_str()	Test Cloudflare's ASN (as a str) '13335' resolves to 'CLOUDFLARENET - Cloudflare, Inc., US'
test_get_rdns_invalid_domain()	Test *get_rdns()* raises `InvalidHost` when given a non-existent domain
test_get_rdns_multi()	Test *get_rdns_multi()* with 3x IPv4 addresses and 1x IPv6 address
test_get_rdns_multi_invalid()	Test *get_rdns_multi()* with 2x IPv4 addresses + 2x IPv6 addresses with one of each having no records
test_get_rdns_no_rdns_records()	Test *get_rdns()* raises `ReverseDNSNotFound` when given a valid IP that has no rDNS records
test_get_rdns_privex_ns1_host()	Test resolving rDNS for the domains `steemseed-fin.privex.io` and `ns1.privex.io`
test_get_rdns_privex_ns1_ip()	Test resolving IPv4 and IPv6 addresses into ns1.privex.io
test_ping()	Test success & failure cases for ping function with IPv4, as well as input validation
test_ping_v6()	Test success & failure cases for ping function with IPv6, as well as input validation

3.20.13.1.1.1 test_asn_to_name_erroneous

TestNet.**test_asn_to_name_erroneous**()

Test asn_to_name returns 'Unknown ASN' when quiet, otherwise throws a KeyError for ASN 'nonexistent'

3.20.13.1.1.2 test_asn_to_name_erroneous_2

TestNet.**test_asn_to_name_erroneous_2**()
Test asn_to_name returns 'Unknown ASN' when quiet, otherwise throws KeyError for the ASN 999999999

3.20.13.1.1.3 test_asn_to_name_int

TestNet.**test_asn_to_name_int**()
Test Privex's ASN (as an int) 210083 resolves to 'PRIVEX, SE'

3.20.13.1.1.4 test_asn_to_name_str

TestNet.**test_asn_to_name_str**()
Test Cloudflare's ASN (as a str) '13335' resolves to 'CLOUDFLARENET - Cloudflare, Inc., US'

3.20.13.1.1.5 test_get_rdns_invalid_domain

TestNet.**test_get_rdns_invalid_domain**()
Test *get_rdns()* raises InvalidHost when given a non-existent domain

3.20.13.1.1.6 test_get_rdns_multi

TestNet.**test_get_rdns_multi**()
Test *get_rdns_multi()* with 3x IPv4 addresses and 1x IPv6 address

3.20.13.1.1.7 test_get_rdns_multi_invalid

TestNet.**test_get_rdns_multi_invalid**()
Test *get_rdns_multi()* with 2x IPv4 addresses + 2x IPv6 addresses with one of each having no records

3.20.13.1.1.8 test_get_rdns_no_rdns_records

TestNet.**test_get_rdns_no_rdns_records**()
Test *get_rdns()* raises ReverseDNSNotFound when given a valid IP that has no rDNS records

3.20.13.1.1.9 test_get_rdns_privex_ns1_host

TestNet.**test_get_rdns_privex_ns1_host**()
Test resolving rDNS for the domains steemseed-fin.privex.io and ns1.privex.io

3.20.13.1.1.10 test_get_rdns_privex_ns1_ip

TestNet.**test_get_rdns_privex_ns1_ip**()
Test resolving IPv4 and IPv6 addresses into ns1.privex.io

3.20.13.1.1.11 test_ping

TestNet.**test_ping**()
Test success & failure cases for ping function with IPv4, as well as input validation

3.20.13.1.1.12 test_ping_v6

TestNet.**test_ping_v6**()
Test success & failure cases for ping function with IPv6, as well as input validation

3.20.13.1.2 Attributes

Attributes

—

3.20.13.2 TestNetResolveIP

class tests.test_net.**TestNetResolveIP**(*methodName='runTest'*)
Test cases related to *resolve_ips()*, *resolve_ip()* and *resolve_ips_multi()*

__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.

3.20.13.2.1 Methods

Methods

test_resolve_ip_hiveseed()	Test *resolve_ip()* returns expected either correct v4 or v6 for hiveseed-fin.privex.io
test_resolve_ip_hiveseed_v4()	Test *resolve_ip()* returns only v4 for hiveseed-fin.privex.io when version is v4
test_resolve_ip_hiveseed_v6()	Test *resolve_ip()* returns only v6 for hiveseed-fin.privex.io when version is v6
test_resolve_ip_v4_convert()	Test *resolve_ip()* returns an IPv6-wrapped IPv4 address for microsoft.com when v4_convert is enabled + v6 version
test_resolve_ips_hiveseed()	Test *resolve_ips()* returns expected v4 + v6 for hiveseed-fin.privex.io

continues on next page

Table 95 – continued from previous page

test_resolve_ips_hiveseed_v4()	Test *resolve_ips()* returns only v4 for `hiveseed-fin.privex.io` when version is set to v4
test_resolve_ips_hiveseed_v6()	Test *resolve_ips()* returns only v6 for `hiveseed-fin.privex.io` when version is set to v6
test_resolve_ips_ipv4_addr()	Test *resolve_ips()* returns the same IPv4 address passed to it
test_resolve_ips_ipv4_addr_invalid()	Test *resolve_ips()* raises `AttributeError` when `version` is v4 but an IPv6 address was passed
test_resolve_ips_ipv6_addr()	Test *resolve_ips()* returns the same IPv6 address passed to it
test_resolve_ips_ipv6_addr_invalid()	Test *resolve_ips()* raises `AttributeError` when `version` is v6 but an IPv4 address was passed
test_resolve_ips_multi_any()	Test *resolve_ips_multi()* with 2 domains and an IPv4 address
test_resolve_ips_multi_v4()	Test *resolve_ips_multi()* with 2 domains, an IPv4 address, and an IPv6 address with version `v4`
test_resolve_ips_multi_v6()	Test *resolve_ips_multi()* with 2 domains, an IPv4 address, and an IPv6 address with version `v6`
test_resolve_ips_v4_convert()	Test *resolve_ips()* returns IPv6-wrapped IPv4 addresses for `microsoft.com` when v4_convert is enabled + v6 version
test_resolve_ips_v4_convert_false()	Test *resolve_ips()* returns an empty list for `microsoft.com` when v6 requested without v4_convert

3.20.13.2.1.1 test_resolve_ip_hiveseed

`TestNetResolveIP.`**`test_resolve_ip_hiveseed`**`()`
Test *resolve_ip()* returns expected either correct v4 or v6 for `hiveseed-fin.privex.io`

3.20.13.2.1.2 test_resolve_ip_hiveseed_v4

`TestNetResolveIP.`**`test_resolve_ip_hiveseed_v4`**`()`
Test *resolve_ip()* returns only v4 for `hiveseed-fin.privex.io` when version is v4

3.20.13.2.1.3 test_resolve_ip_hiveseed_v6

`TestNetResolveIP.`**`test_resolve_ip_hiveseed_v6`**`()`
Test *resolve_ip()* returns only v6 for `hiveseed-fin.privex.io` when version is v6

3.20.13.2.1.4 test_resolve_ip_v4_convert

TestNetResolveIP.**test_resolve_ip_v4_convert**()
Test *resolve_ip()* returns an IPv6-wrapped IPv4 address for `microsoft.com` when v4_convert is enabled + v6 version

3.20.13.2.1.5 test_resolve_ips_hiveseed

TestNetResolveIP.**test_resolve_ips_hiveseed**()
Test *resolve_ips()* returns expected v4 + v6 for `hiveseed-fin.privex.io`

3.20.13.2.1.6 test_resolve_ips_hiveseed_v4

TestNetResolveIP.**test_resolve_ips_hiveseed_v4**()
Test *resolve_ips()* returns only v4 for `hiveseed-fin.privex.io` when version is set to v4

3.20.13.2.1.7 test_resolve_ips_hiveseed_v6

TestNetResolveIP.**test_resolve_ips_hiveseed_v6**()
Test *resolve_ips()* returns only v6 for `hiveseed-fin.privex.io` when version is set to v6

3.20.13.2.1.8 test_resolve_ips_ipv4_addr

TestNetResolveIP.**test_resolve_ips_ipv4_addr**()
Test *resolve_ips()* returns the same IPv4 address passed to it

3.20.13.2.1.9 test_resolve_ips_ipv4_addr_invalid

TestNetResolveIP.**test_resolve_ips_ipv4_addr_invalid**()
Test *resolve_ips()* raises `AttributeError` when `version` is v4 but an IPv6 address was passed

3.20.13.2.1.10 test_resolve_ips_ipv6_addr

TestNetResolveIP.**test_resolve_ips_ipv6_addr**()
Test *resolve_ips()* returns the same IPv6 address passed to it

3.20.13.2.1.11 test_resolve_ips_ipv6_addr_invalid

TestNetResolveIP.**test_resolve_ips_ipv6_addr_invalid**()
Test *resolve_ips()* raises `AttributeError` when `version` is v6 but an IPv4 address was passed

3.20.13.2.1.12 test_resolve_ips_multi_any

TestNetResolveIP.**test_resolve_ips_multi_any**()
Test *resolve_ips_multi()* with 2 domains and an IPv4 address

3.20.13.2.1.13 test_resolve_ips_multi_v4

TestNetResolveIP.**test_resolve_ips_multi_v4**()
Test *resolve_ips_multi()* with 2 domains, an IPv4 address, and an IPv6 address with version `v4`

3.20.13.2.1.14 test_resolve_ips_multi_v6

TestNetResolveIP.**test_resolve_ips_multi_v6**()
Test *resolve_ips_multi()* with 2 domains, an IPv4 address, and an IPv6 address with version `v6`

3.20.13.2.1.15 test_resolve_ips_v4_convert

TestNetResolveIP.**test_resolve_ips_v4_convert**()
Test *resolve_ips()* returns IPv6-wrapped IPv4 addresses for `microsoft.com` when v4_convert is enabled + v6 version

3.20.13.2.1.16 test_resolve_ips_v4_convert_false

TestNetResolveIP.**test_resolve_ips_v4_convert_false**()
Test *resolve_ips()* returns an empty list for `microsoft.com` when v6 requested without v4_convert

3.20.13.2.2 Attributes

Attributes

—

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

t

INDEX

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U