prybar Documentation

Release 1.0.0b2

Hal Blackburn

Apr 07, 2019

Contents

Use prybar to temporarily define `pkg_resources` entry points at runtime. The primary use case is testing code which works with entry points.

CHAPTER 1

Entry points?

Entry points are Python's way of advertising and consuming plugins. Python packages can advertise an object (function, class, data, etc) which can be discovered and loaded by another package.

Entry points are normally statically defined in package metadata (e.g. via `setup.py`). The static nature of entry points makes thoroughly testing code that loads entry points awkward.

prybar allows entry points to be created and removed on the fly, making it easy to test your plugin loading code.

CHAPTER 2

Installing

prybar is available from PyPi as `prybar`:

```
$ pip install prybar
```

prybar requires Python 3.6 or greater.

CHAPTER 3

Using prybar

prybar provides `dynamic_entrypoint()` — a context manager which creates an entry point when entered and removes it when left. It can also be used as a function decorator, or via explicit `.start()` and `.stop()` calls.

3.1 Brief Example

`pkg_resources.iter_entry_points` is the normal way to discover entry points. We'll define a function to load the first named entry point in a group (category).

```
>>> from pkg_resources import iter_entry_points
>>> def load_entrypoint(group, name):
...     return next((ep.load() for ep in
...                  iter_entry_points(group, name=name)), None)
```

Initially no entry point will exist:

```
>>> load_entrypoint('example.hash_types', 'sha256') is None
True
```

We can create it at runtime with prybar:

```
>>> from prybar import dynamic_entrypoint
>>> with dynamic_entrypoint('example.hash_types',
...                         name='sha256', module='hashlib'):
...     hash = load_entrypoint('example.hash_types', 'sha256')
...     hash(b'foo').hexdigest()[:6]
'2c26b4'
```

It's gone again after leaving the `with` block:

```
>>> load_entrypoint('example.hash_types', 'sha256') is None
True
```

3.2 More examples

Multiple entry points can be registered by nesting `with` blocks, or using `contextlib.ExitStack`:

```
>>> from contextlib import ExitStack
>>> epoints = [dynamic_entrypoint('example.types',
...                               name=name, module='builtins')
...            for name in ['int', 'float', 'str', 'bool']]
>>> with ExitStack() as stack:
...     for ep in epoints:
...         stack.enter_context(ep)
...
...     for ep in iter_entry_points('example.types'):
...         t = ep.load()
...         t('12')
12
12.0
'12'
True
```

`dynamic_entrypoint` can also be used as a decorator:

```
>>> @dynamic_entrypoint('example.hash_types',
...                     name='sha256', module='hashlib')
... def example_function():
...     hash = load_entrypoint('example.hash_types', 'sha256')
...     return hash(b'foo').hexdigest()[:6]
>>> load_entrypoint('example.hash_types', 'sha256') is None
True
>>> example_function()
'2c26b4'
```

And via `start()` and `stop()` methods:

```
>>> sha256_dep = dynamic_entrypoint(
...     'example.hash_types', name='sha256', module='hashlib')
>>> load_entrypoint('example.hash_types', 'sha256') is None
True
>>> sha256_dep.start()
>>> hash = load_entrypoint('example.hash_types', 'sha256')
>>> hash(b'foo').hexdigest()[:6]
'2c26b4'
>>> sha256_dep.stop()
>>> load_entrypoint('example.hash_types', 'sha256') is None
True
```

The entry point can be specified in several ways in addition to the `name` and `module` seen above.

`attribute` can be used if the desired entry point name differs from the target's name in the module:

```
>>> from collections import Counter
>>> with dynamic_entrypoint('example', name='thing',
...                         module='collections',
...                         attribute='Counter'):
...     thing = load_entrypoint('example', 'thing')
...     thing is Counter
True
```

A function or class can be passed as `entrypoint`, from which the `name`, `module` and `attribute` are inferred:

```
>>> with dynamic_entrypoint('example', entrypoint=Counter):
...     thing = load_entrypoint('example', 'Counter')
...     thing is Counter
True
```

The `name` attribute can be used to override the entry point's name:

```
>>> with dynamic_entrypoint('example', name='foo',
...                         entrypoint=Counter):
...     thing = load_entrypoint('example', 'foo')
...     thing == Counter
True
```

Nested functions can be specified:

```
>>> with dynamic_entrypoint('example', module='collections',
...                         name='fromkeys',
...                         attribute=('Counter', 'fromkeys')):
...     thing = load_entrypoint('example', 'fromkeys')
...     thing == Counter.fromkeys
True
>>> with dynamic_entrypoint('example', entrypoint=Counter.fromkeys):
...     thing = load_entrypoint('example', 'fromkeys')
...     thing == Counter.fromkeys
True
>>> with dynamic_entrypoint('example', name='foo',
...                         entrypoint=Counter.fromkeys):
...     thing = load_entrypoint('example', 'foo')
...     thing == Counter.fromkeys
True
```

A string using the entry point syntax from `setup.py` can be used:

```
>>> with dynamic_entrypoint(
...         'example',
...         entrypoint='thing = collections:Counter.fromkeys'):
...     thing = load_entrypoint('example', 'thing')
...     thing == Counter.fromkeys
True
```

Or a `pkg_resources.EntryPoint` object can be passed:

```
>>> from pkg_resources import EntryPoint
>>> ep = EntryPoint('thing', 'collections', attrs=('Counter', 'fromkeys'))
>>> with dynamic_entrypoint('example', entrypoint=ep):
...     thing = load_entrypoint('example', 'thing')
...     thing == Counter.fromkeys
True
```

CHAPTER 4

API Reference

4.1 *prybar*