bender-hooks Documentation

Release 0.1.0

Bruno Oliveira, Fabio Menegazzo

December 11, 2014

Contents

bender-hooks is a library which provides ways to easily create, search and invoke hooks anywhere in your code.

Hooks will be created as decorators so all you have to do is decorating functions you want to be invoked through that hook.

It can be used to define interfaces or callbacks for your application.

CHAPTER 1

Install

To install **bender-hooks** all it takes is one command line:

```
pip install bender-hooks
```

CHAPTER 2

Documentation

2.1 Getting started

First of all you have to create your hook signature. It is much like an interface definition where you only have to define the parameters and related documentation.

definitions.py:

```
def grettings(greet, name):
    """
    Called in scripts that want to print greetings.

    :param greet: unicode
    :param name: unicode
    """

def goodbye():
    """
    Called in scripts that want to print goodbye messages.
    """
```

Now you can turn your definition into a decorator using **bender-hooks**. This way your decorator will work much like an interface.

decorators.py:

```
import bender_hooks
import definitions
greetings = bender_hooks.make_decorator(definitions.greetings)
goodbye = bender_hooks.make_decorator(definitions.goodbye)
```

With decorator in hands you just have to mark the functions that will be invoked through the hook. The defined parameters are not mandatory on such functions.

hello.py:

```
import decorators

@decorators.greetings
def a(greet, name):
    print("%s, %s!" % (greet, name))

@decorators.greetings
def b(name):
```

```
    print("Hi, %s!" % name)

@decorators.greetings
def c():
    print("Hi there!")

@decorators.goodbye
def d():
    print("Farewell!")
```

As all *grettings* functions are already marked, just ask **bender-hooks** to call them:

```
>>> import bender_hooks
>>> import hello
>>> bender_hooks.call_all_hooks(hello, 'greetings', greet='Welcome', name='John')
Welcome, John!
Hi, John!
Hi there!
>>> bender_hooks.call_all_hooks(hello, 'goodbye')
Farewell!
```

Sometimes it is necessary having just one implementation for a given hook. To make sure of this just call:

```
>>> bender_hooks.call_unique_hook(hello, 'greetings', greet='Welcome', name='John')
bender_hooks.HookError: <module 'hello' from 'hello.pyc'> can implement greetings at most one time
>>> bender_hooks.call_unique_hook(hello, 'goodbye')
Farewell!
```

This is the basics! For more detailed information please refer to API documentation.

2.2 API

This document brings the public API of **bender-hooks**.

exception `bender_hooks.HookError`
Error raised for expected and known cases of **bender-hooks**.

`bender_hooks.call`(*hook*, ***kwargs*)
Responsible for invoking given `hook`. As hooks not necessarily defines all arguments, this function will make sure that the given one will satisfy the `hook`. Example:

```
def foo(a, b):
    """
    My decorator definition.
    """

# Creating decorator.
dec = bender_hooks.make_decorator(foo)

# Decorating. Only one parameter defined.
@foo
def bar(b):
    print(b)

# Invoking.
bender_hooks.call(bar, a=1) # Invalid: ''a'' is not defined at ''bar''
bender_hooks.call(bar, b=2) # Valid
bender_hooks.call(bar, c=3) # Invalid: ''c'' is not defined at ''bar'' neither ''foo''
```

```
bender_hooks.call(bar, a=1, b=2) # Valid: ``a`` is defined at ``foo`` but will be ignored.
bender_hooks.call(bar, b=2, c=3) # Invalid
bender_hooks.call(bar, a=1, b=2, c=3) # Invalid
```

Parameters **hook** (*callable*) – Already decorated function.

Returns It depends on what will be returned by given `hook`.

Raises **HookError** if given `hook` is not decorated.

`bender_hooks.`**`call_all_hooks`**(*obj*, *hook_name*, ***kwargs*)

Responsible for search and invoke all hooks with the given `hook_name` under `obj`.

See also:

`find_hooks()`

See also:

`call()`

`bender_hooks.`**`call_unique_hook`**(*obj*, *hook_name*, ***kwargs*)

Responsible for search and invoke a hook with the given `hook_name` under `obj`, making sure that only one hook exists.

Raises **HookError** if more than one hooks are found.

See also:

`find_hooks()`

See also:

`call()`

`bender_hooks.`**`find_hooks`**(*obj*, *hook_name*)

Responsible for search at `obj` hooks with the given `hook_name`.

Parameters

- **obj** – Object (can be a module or instance) to search for hooks.
- **hook_name** (*unicode*) – Name of hook to search.

Return type list(callable)

Returns All hooks found into the given `obj`.

`bender_hooks.`**`make_decorator`**(*hook_decl*, *inputs=()*)

Responsible for turning a hook declaration into a decorator.

Parameters

- **hook_decl** (*callable*) – The definition that will be turn into a decorator. Example:

```
def grettings(greet, name):
    """
    Called in scripts that want to print greetings.

    :param greet: unicode
    :param name: unicode
    """
```

If any code is placed inside the given definition it will be ignored.

- **inputs** (*tuple* | *list*) – Parameters that must be given to decorator. Example:

```
dec = bender_hooks.make_decorator(foo, inputs=('alpha', 'bravo'))

@dec('A', 'B')
def my_func():
    pass

my_func.inputs['alpha'] == 'A'
my_func.inputs['bravo'] == 'B'
```

Return type callable

Returns The created decorator.

Raises HookError when given arguments are not valid.

CHAPTER 3

Support

Need help? Click here and open a new issue. You'll get your answer ASAP.

CHAPTER 4

Contribute

bender-hooks is under development, so if you want to join the team, you are welcome.

- Feel free to open issues related to bugs or ideas.
- If you are a developer:
  - Fork **bender-hooks** before making any changes.
  - Write tests.
  - Create a *Pull Request* so changes can be merged.

CHAPTER 5

License

bender-hooks is licensed under LGPL v3.0 license.

b

B

C

F

H

M