Shout

Release 0.2.3

Mar 05, 2018

Contents

1 Why Shout **3**

2 Get Shout **5**

3 Index **7**

3.1 Guide 7

3.2 API Documentation 9

Loud Python messaging.

Shout is a single module providing simple messaging vocabulary for small applications. Shout is NOT a distributed messaging framework.

```
from shout import Message, hears, shout

class WhoAreYou(Message):
    pass

@hears(WhoAreYou)
def lucky_day():
    return "We are..."

@hears(WhoAreYou)
def dusty_bottoms():
    return "The threeee..."

@hears(WhoAreYou)
def ned_nederlander():
    return "Amigos!!"

msg = shout(WhoAreYou)
print("".join(msg.results))

# We are...The threeee...Amigos!!
```

CHAPTER 1

Why Shout

- Decoupling of a GUI and it's behavior
  - PySide/PyQt signals are bound to widgets making it harder to decouple widgets. You have to explicitly connect each widget's signals with their slots which could live deep in a hierarchy of widgets.
  - Shout Messages are classes themselves, readily available to all other objects in their scope. Shout from inside, outside, top, or bottom of a widget hierarchy, Messages will still get to where they need to go!
- Shout is a single module, easily included with any package.
- It's easy and fun to use.

CHAPTER 2

Get Shout

Shout is available through the python package index as **pyshout**.

```
pip install pyshout
```

- Note that only the python package name is pyshout, the module it installs is simply **shout**.

CHAPTER 3

Index

3.1 Guide

This section provides everything you need to know about using Shout.

3.1.1 Creating a Message

Start by importing the essentials from shout.

```
from shout import Message, has_ears, hears, shout
```

Now we can create a new type of *Message*.

```
class MyMessage(Message):
    pass
```

Our *Message* type will allow us to *shout()* args and kwargs around our application. But, before we can do that...

3.1.2 Who is Listening?

Let's make a function that can actually hear us *shout()* our *Message* s.

```
@hears(MyMessage, inside="A")
def maximum(msg):
    return upper(msg) + "!!"
```

`maximum()` will hear all `MyMessage` shouts inside room "A". In this case only one type of *Message* will be heard, but multiple *Message* s can be passed as args to *hears()*. Additionally multiple rooms can be passed as a tuple to the inside keyword argument. If you don't pass any room names to inside, your function will listen in the default room, "void".

3.1.3 Does your class have ears?

You're every day class doesn't have ears so it's methods won't be able to hear any shouted *Message* s. It's **super** simple to give a class ears, just decorate it with *has_ears()*!

```
@has_ears
class Volumes(object):

    @hears(MyMessage)
    def low(msg):
        return lower(msg)

    @hears(MyMessage)
    def medium(msg):
        return msg.title()

    @hears(MyMessage)
    def high(msg):
        return upper(msg)

v = Volumes()
```

Once we've given our class ears, the last thing we have to do is create an instance of it. On instantiation the bound methods are added as listeners to the appropriate *Message* s.

3.1.4 Shout at the top of your lungs!

We've got our *Message* and a bunch of listeners, now we can shout all we want to.

```
m = shout(MyMessage, "hello there", inside="A")
```

Now we've shouted a *Message* and we've got a *Message* instance bound to **m**. *Message* instances have a bunch of useful attributes.

```
print "args, kwargs: ", m.args, m.kwargs
print "response: ", m.response
print "success: ", m.success
print "exception: ", m.exc

# args, kwargs: ("hello there", ), {}
# response: ["HELLO THERE!!"]
# success: True
# exception: None
```

Cool, but, judging from the response, none of our methods in `Volumes` heard us shout. That's because we shouted inside room "A". Let's see what happens if we shout again but this time, not explicitly passing a room to the **inside** keyword.

```
m = shout(MyMessage, "hello again")

print "args, kwargs: ", m.args, m.kwargs
print "response: ", m.response
print "success: ", m.success
print "exception: ", m.exc

# args, kwargs: ("hello again", ), {}
# response: ["hello again", "Hello Again", "HELLO AGAIN"]
```

```
# success: True
# exception: None
```

There we go! This time we've shouted inside the default room "void", reaching all of our `Volumes` instance's listeners. It's important to note that while we only passed one argument in our shouts, any arg, kwarg signature is supported. *`Message`* signatures should be set by their listeners. So, if you have multiple listeners for the same type of *`Message`*, ensure that they all take the same parameters.

3.1.5 Debugging

Shout has extensive logging which is turned off by default.

```
import logging
shout_logger = logging.getLogger('Shout!')
shout_logger.setLevel(logging.DEBUG)
```

This will set Shouts logger level to logging.DEBUG. Printing out a ton of useful messages! You can also log to a file.

3.2 API Documentation

3.2.1 Message

class shout.**Message**(**args*, ***kwargs*)

`Message` s keep track of their listeners and the various rooms they are listening to. Instances of *`Message`* hold args and kwargs and when *`shout()`* is called these are passed to all the appropraite listeners. All return values of listeners are collected in `response`. If all listeners execute correctly `success` is set to True. Any Exception raised by a listener will halt the shout after binding `exc` to the offending Exception.

Parameters

- **args** – Arguments to shout
- **kwargs** – Keyword Arguments to shout

static create(*name*)

Dynamically create a new type of *`Message`*.

Parameters **name** – The __class__.__name__ to use.

shout()

Sends the instances args and kwargs to the appropriate listeners.

3.2.2 hears

shout.**hears**(**args*, ***kwargs*)

Decorates functions and methods, adding them as listeners to the specified *`Message`* s.

Parameters

- **args** – *`Message`* s this function will hear.
- **inside** – Tuple of rooms this function will hear.

3.2.3 has_ears

shout.**has_ears**(*cls*)
Class decorator that enables *hears()* decorator to be used on class methods.

3.2.4 shout

shout.**shout**(*msg_type*, **args*, ***kwargs*)
A grammatically pleasant way to shout a *Message*.

shout(Message, “Hello”, inside=”A”) == Message(“Hello”, inside=”A”).shout()

Parameters

- **msg_type** – The type of *Message* to shout.
- **args** – The args to pass to the *Message*.
- **kwargs** – The kwargs to pass to the *Message*.
- **inside** – The rooms to shout inside.

3.2.5 shout_logging

Index

C

create() (shout.Message static method), 9

H

has_ears() (in module shout), 10
hears() (in module shout), 9

M

Message (class in shout), 9

S

shout() (in module shout), 10
shout() (shout.Message method), 9