Ether documentation

version 0.2.0

Islam Amer

Ed Bartosh

Anton Beresin

March 01, 2011

Contents

Welcome to Ether's documentation!

Contents:

Overview

This project is an implementation of commit / post-recieve hooks for various VCS systems, that send a notification to an AMQP server containing payload that follows the github payload spec :

Development practices

General ideas

Code in this project should satisfy following requirements:

- PEP8 compliant
- Covered with testcases not less than 90%
- Covered with documentation
- Reviewed

Code Quality

We use pylint with default settings to check the code compliance:

```
pylint <your-file>
```

Code rated < 9 is **NOT** acceptable.

Before you commit your changes please run the unit tests

Make sure that all the test cases pass and code coverage is more than 90%.

After you fix a defect please create a test case to avoid regressions in future.

Documenting your code

For documenting we use sphinx You can find project documentation in docs directory of the project

Here are recomendations of documenting API:

If you document your functions this way:

```
def foo(param1, param2):
    """Description of foo.

    :param param1: param1 description
    :type param1: string

    :param param2: param2 description
    :type param2: list

    :returns: tuple (<category name>, <list of items>)
    :rtype: tuple

    """

    # some code here
    return (category, items)
```

you'll have this nice documentation generated by *sphinx*:

`docfunc.`**`foo`** (*param1*, *param2*)
Description of foo.

Parameters:

- **param1** (*string*) -- param1 description
- **param2** (*list*) -- param2 description

Returns: tuple (<category name>, <list of items>)

Return type: tuple

The same is for classes:

```
class Foo(object):
    """Class description."""

    #: class attribute description.
    classattr = None

    def method(self, param1, param2):
        """
        Method description.
        Could take several lines.

        :param param1: param1 description
        :type param1: string
        :param param2: param2 description
        :type param2: list

        """

        pass
```

class `docclass.`**`Foo`**
Class description.

`classattr`
class attribute description.

`method` **(*param1*, *param2*)**
Method description. Could take several lines.

Parameters:

- **param1** (*string*) -- param1 description
- **param2** (*list*) -- param2 description

Some useful links related to sphinx:

- http://docutils.sourceforge.net/docs/user/rst/quickref.html
- http://sphinx.pocoo.org/markup/code.html
- http://sphinx.pocoo.org/ext/autodoc.html
- http://packages.python.org/an_example_pypi_project/sphinx.html

Branching policy

Recommended way is to have separate git branch for each task. After code is ready branch can be rebased against master and provided for review.

Please note that **No code should go to master without review** It's reviewer task to review the code, disscuss it with developer, then merge it to master and tag it if needed.

Building Debian Package

Use:

```
git-buildpackage -rfakeroot -uc -us -sa -D
```

to build packages.

Build debian package is a recommended way to check your work, because project documentation is re-generated and unit tests are run during package building.

Modules and classes

Contents:

Senders

amqp

AMQP sender API.

class ether.publishers.amqp.**BasicAMQPPublisher** (*config={'PUBLISHER': {'queue_durable': True, 'queue_auto_delete': False, 'queue_name': '', 'routing_key': '', 'queue_exclusive': False}, 'exchange_name': 'ether', 'delivery_mode': 1, 'vhost': '/ether', 'host': 'localhost', 'user': 'ether', 'exchange_durable': True, 'port': 5672, 'password': '123', 'exchange_type': 'fanout', 'CONSUMER': {'queue_durable': True, 'queue_auto_delete': True, 'queue_name': '', 'routing_key': '', 'queue_exclusive': True}}*)

Base abstract class for AMQP publishers.

send_payload (***payload***)

Send payload to the server. Abstract method. Has to be implemented in derived classes.

Parameters: **payload** (*dictionary*) -- data to be sent

class ether.publishers.amqp.**BlockingAMQPPublisher** (*config={'PUBLISHER': {'queue_durable': True, 'queue_auto_delete': False, 'queue_name': '', 'routing_key': '', 'queue_exclusive': False}, 'exchange_name': 'ether', 'delivery_mode': 1, 'vhost': '/ether', 'host': 'localhost', 'user': 'ether', 'exchange_durable': True, 'port': 5672, 'password': '123', 'exchange_type': 'fanout', 'CONSUMER': {'queue_durable': True, 'queue_auto_delete': True, 'queue_name': '', 'routing_key': '', 'queue_exclusive': True}}*)

Blocking (synchronous) publisher. Code is borrowed from Pika Blocking demo_send example_blocking:

send_payload (***payload***)

Send payload to the server using blocking approach.

Parameters: **payload** (*dictionary*) -- data to be sent

class ether.publishers.amqp.**AsyncAMQPPublisher** (*config={'PUBLISHER': {'queue_durable': True, 'queue_auto_delete': False, 'queue_name': '', 'routing_key': '', 'queue_exclusive': False}, 'exchange_name': 'ether', 'delivery_mode': 1, 'vhost': '/ether', 'host': 'localhost', 'user': 'ether', 'exchange_durable': True, 'port': 5672, 'password': '123', 'exchange_type':*

'fanout', 'CONSUMER': {'queue_durable': True, 'queue_auto_delete': True, 'queue_name': '', 'routing_key': '', 'queue_exclusive': True}})

Asynchronous Publisher. Code is borrowed from Pika Asynchronous demo_send example_async:

Methods are placed in the same order as they're called by pika

`on_channel_open` (***`channel`***)

Callback. Called when channel has opened.

Parameters: **channel** (*object*) -- channel object

`on_connected` (***`connection`***)

Callback. Called when we are fully connected to RabbitMQ.

Parameters: **connection** (*object*) -- connection object

`on_queue_declared` (***`_frame`***)

Callback: Called when queue has been declared.

Parameters: **_frame** (*object*) -- response from broker

`send_payload` (***`payload`***)

Send payload to the server setting up chain of callbacks: on_connected -> on_channel_open -> on_queue_declared. (see above)

Parameters: **payload** (*dictionary*) -- data to be sent

log

class `ether.publishers.log.`**`FileLogger`** (*`filename`*)

`send_payload` (***`payload`***)

Listeners

amqp

AMQP Listener.

class `ether.consumers.amqp.`**`BaseAMQPConsumer`** (*config={'PUBLISHER': {'queue_durable': True, 'queue_auto_delete': False, 'queue_name': '', 'routing_key': '', 'queue_exclusive': False}, 'exchange_name': 'ether', 'delivery_mode': 1, 'vhost': '/ether', 'host': 'localhost', 'user': 'ether', 'exchange_durable': True, 'port': 5672, 'password': '123', 'exchange_type': 'fanout', 'CONSUMER': {'queue_durable': True, 'queue_auto_delete': True, 'queue_name': '', 'routing_key': '', 'queue_exclusive': True}}*)

Base abstract class for AMQP consumers.

`receive_payload` (***`channel`***, ***`method`***, ***`header`***, ***`body`***)

Recieve payload from the server. Abstract method. Has to be implemented in derived classes.

Parameters: **body** (*dictionary*) -- data received

class `ether.consumers.amqp.`**`AsyncAMQPConsumer`** (*config={'PUBLISHER': {'queue_durable': True, 'queue_auto_delete': False, 'queue_name': '', 'routing_key': '', 'queue_exclusive': False}, 'exchange_name': 'ether', 'delivery_mode': 1, 'vhost': '/ether', 'host': 'localhost', 'user': 'ether', 'exchange_durable': True, 'port': 5672, 'password': '123', 'exchange_type': 'fanout', 'CONSUMER': {'queue_durable': True, 'queue_auto_delete': True,*

'queue_name': '', 'routing_key': '', 'queue_exclusive': True}})
Asynchronous consumer.

`consume ()`
Start the IO event loop so we can communicate with RabbitMQ.

`on_channel_open (`*`channel`*`)`
Step #3: Called when our channel has opened.

Parameters: **channel** (*object*) -- channel object

`on_connected (`*`connection`*`)`
Step #2: Called when we are fully connected to RabbitMQ.

Parameters: **connection** (*object*) -- connection object

`on_queue_bound (`*`frame`*`)`
Step #6: Called when the queue has been bound to the exchange.

Parameters: **_frame** (*object*) -- response from broker

`on_queue_declared (`*`frame`*`)`
Step #5: Called when Queue has been declared.
frame is the response from RabbitMQ

`receive_payload (`*`channel`*`, `*`method`*`, `*`header`*`, `*`body`*`)`
Step #7: Called when we receive a message from RabbitMQ. Implementation of recieve_payload that just prints the payload.

Parameters: **body** (*dictionary*) -- data received

`setup_connection ()`
Step #1: Connect to RabbitMQ.

Hooks

Contents:

Indices and tables

- *genindex*
- *modindex*
- *search*

Indices and tables

- *genindex*
- *modindex*
- *search*

Index

Python Module Index