bearychat.py Documentation

Release 0.0.1

Beary Innovative

May 15, 2017

Contents:

BearyChat.py is a SDK for BearyChat.

CHAPTER 1

Quick links

- Source(GitHub)
- BearyChat Integrations

CHAPTER 2

Hello, world

Here is a simple "Hello, world" example app for BearyChat Incoming:

```
from bearychat import incoming

def main():
    data = {
        "text": "hello, **world**",
        "markdown": True,
        "notification": "Hello, BearyChat in Notification",
        "channel": "testing"
    }

    resp = incoming.send(
        "https://hook.bearychat.com/=bw52O/incoming/token",
        data)

    print(resp.status_code)
    print(resp.text)


if __name__ == "__main__":
    main()
```

CHAPTER 3

Installation

Automatic installation:

```
pip install bearychat
```

BearyChat.py is listed in PyPI, and can be installed with `pip` or `easy_install`.

Prerequisites: BearyChat.py runs on Python 2.6+ and Python 3.3+(more). And HTTP library request is required.

CHAPTER 4

Documentation

User's guide

Introduction

BearyChat.py is a SDK for BearyChat

Incoming

Incoming is an integration of BearyChat.

Examples

Here is a simple incoming workflow:

```
from bearychat import incoming

data = {
    "text": "hello, **world**",
    "markdown": True,
    "notification": "Hello, BearyChat in Notification",
    "channel": "testing"
}

resp = incoming.send(
     "https://hook.bearychat.com/=bw52O/incoming/token",
     data)
```

Real Time Message

RTM Message

Provides handful helpers for rtm message parsing.

RTM Loop

To achive more flexible usage, BearyChat.py won't provide any implementations for rtm.loop. You can use examples/rtm_loop below as implementation reference.

Basically, rtm.loop contains 3 stages:

- rtm.start: Use rtm token to authenticate user and open a websocket connection.
- ping: Keep sending type=ping message to server after connected. Pinging interval with 5000ms is suggested.
- loop: Subscribe to websocket's message event and consume the message comes from the server. You can use RTMMessage for message parsing.

Examples

Here is a sample rtm loop implementation:

```
import sys
import time
import json
import threading

import websocket

from bearychat import RTMMessage, RTMMessageType

if sys.version_info > (3, ):
    from queue import Queue
    from _thread import start_new_thread
else:
    from Queue import Queue
    from thread import start_new_thread


class RTMLoop(object):
    """Real Time Message loop

    _errors(Queue): contains error message(dict("result", "msg")),
                    looks self._set_error()
    _inbox(Queue): contains RTMMessage
    _worker(threading.Thread): a thread for running the loop

    Args:
        ws_host(str): websocket host
    """

    def __init__(self, ws_host):
        self._call_id = 0
        self._inbox = Queue()
        self._errors = Queue()
        self._ws = websocket.WebSocketApp(
            ws_host,
```

```
            on_open=self.on_open,
            on_message=self.on_message,
            on_close=self.on_close,
            on_error=self.on_error)
        self._worker = threading.Thread(target=self._ws.run_forever)

    def on_open(self, ws):
        """Websocket on_open event handler"""
        def keep_alive(interval):
            while True:
                time.sleep(interval)
                self.ping()

        start_new_thread(keep_alive, (self.keep_alive_interval, ))

    def on_message(self, ws, message):
        """Websocket on_message event handler

        Saves message as RTMMessage in self._inbox
        """
        try:
            data = json.loads(message)
        except:
            self._set_error(message, "decode message failed")
        else:
            self._inbox.put(RTMMessage(data))

    def on_error(self, ws, error):
        """Websocket on_error event handler

        Saves error message in self._errors
        """
        self._set_error(error, "read socket failed")

    def on_close(self, ws):
        """Websocket on_close event handler"""
        self._set_error("closed", "websocket closed")

    def _set_error(self, result, msg):
        """Puts a error to self._errors

        Args:
            result(mix): received data
            msg(str): message
        """
        self._errors.put({"result": result, "msg": msg})

    def start(self, keep_alive_interval=2):
        """Starts the main loop

        Args:
            keep_alive_interval(int): the interval(second) of sending keep
                                      alive message
        """
        self.keep_alive_interval = keep_alive_interval
        self._worker.start()

    def stop(self):
```

```
        """Stops the main loop
        """
        self._ws.close()

    def ping(self):
        """Sends ping message
        """
        self.send(RTMMessage({"type": RTMMessageType.Ping}))

    def gen_call_id(self):
        """Generates a call_id

        Returns:
            int: the call_id
        """
        self._call_id += 1
        return self._call_id

    def send(self, message):
        """Sends a RTMMessage
        Should be called after starting the loop

        Args:
            message(RTMMessage): the sending message

        Raises:
            WebSocketConnectionClosedException: if the loop is closed
        """
        if "call_id" not in message:
            message["call_id"] = self.gen_call_id()

        self._ws.send(message.to_json())

    def get_message(self, block=False, timeout=None):
        """Removes and returns a RTMMessage from self._inbox

        Args:
            block(bool): if True block until a RTMMessage is available,
                         else it will return None when self._inbox is empty
            timeout(int): it blocks at most timeout seconds

        Returns:
            RTMMessage if self._inbox is not empty, else None
        """
        try:
            message = self._inbox.get(block=block, timeout=timeout)
            return message
        except:
            return None

    def get_error(self, block=False, timeout=None):
        """Removes and returns an error from self._errors

        Args:
            block(bool): if True block until a RTMMessage is available,
                         else it will return None when self._inbox is empty
            timeout(int): it blocks at most timeout seconds
```

```
        Returns:
            error if inbox is not empty, else None
        """
        try:
            error = self._errors.get(block=block, timeout=timeout)
            return error
        except:
            return None
```

And Here is the rtm loop above working sample:

```
import time

from bearychat import RTMClient

from rtm_loop import RTMLoop

client = RTMClient("rtm_token", "https://rtm.bearychat.com")
# init the rtm client

resp = client.start()  # get rtm user and ws_host

user = resp["user"]
ws_host = resp["ws_host"]

loop = RTMLoop(ws_host)  # init the loop
loop.start()
time.sleep(2)

while True:
    error = loop.get_error()

    if error:
        print(error)
        continue

    message = loop.get_message(True, 5)

    if not message or not message.is_chat_message():
        continue
    try:
        print("rtm loop received {0} from {1}".format(message["text"],
                                                      message["uid"]))
    except:
        continue

    if message.is_from(user):
        continue
    loop.send(message.refer("Pardon?"))
```

Release Notes

What's new in BearyChat.py 0.3.0

May 15, 2017

OpenAPI

- Adds OpenAPI support.

What's new in BearyChat.py 0.0.2

Dec 30, 2016

Incoming

- Supports creating and sending message to BearyChat Incoming.

Real Time Message

- Supports creating real time message.
- genindex
- modindex
- search

CHAPTER 5

Discussion and support

You can report bugs on the GitHub issue tracker