caronc / apprise Public

Code Issues 41 Pull requests 6 Discussions Actions Wiki

0007eade20

apprise / apprise / plugins / NotifyIFTTT.py / <> Jump to

caronc Strict enforcing of +, -, and : prefixed kwargs in URLs (#302) ✓ History

1 contributor

371 lines (306 sloc) | 12.8 KB

```python
# -*- coding: utf-8 -*-
#
# IFTTT (If-This-Then-That)
#
# Copyright (C) 2019 Chris Caron <lead2gold@gmail.com>
# All rights reserved.
#
# This code is licensed under the MIT License.
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files(the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and / or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions :
#
# The above copyright notice and this permission notice shall be included in
# all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
# THE SOFTWARE.
#
# For this plugin to work, you need to add the Maker applet to your profile
# Simply visit https://ifttt.com/search and search for 'Webhooks'
# Or if you're signed in, click here: https://ifttt.com/maker_webhooks
# and click 'Connect'
#
# You'll want to visit the settings of this Applet and pay attention to the
# URL. For example, it might look like this:
#               https://maker.ifttt.com/use/a3nHB7gA9TfBQSqJAHklod
#
# In the above example a3nHB7gA9TfBQSqJAHklod becomes your {webhook_id}
# You will need this to make this notification work correctly
#
# For each event you create you will assign it a name (this will be known as
# the {event} when building your URL.
import re
import requests
from json import dumps

from .NotifyBase import NotifyBase
from ..common import NotifyType
from ..utils import parse_list
from ..utils import validate_regex
from ..AppriseLocale import gettext_lazy as _


class NotifyIFTTT(NotifyBase):
    """
    A wrapper for IFTTT Notifications

    """

    # The default descriptive name associated with the Notification
    service_name = 'IFTTT'

    # The services URL
    service_url = 'https://ifttt.com/'

    # The default protocol
    secure_protocol = 'ifttt'

    # A URL that takes you to the setup/help of the specific protocol
    setup_url = 'https://github.com/caronc/apprise/wiki/Notify_ifttt'

    # Even though you'll add 'Ingredients' as {{ Value1 }} to your Applets,
    # you must use their lowercase value in the HTTP POST.
    ifttt_default_key_prefix = 'value'

    # The default IFTTT Key to use when mapping the title text to the IFTTT
    # event. The idea here is if someone wants to over-ride the default and
    # change it to another Ingredient Name (in 2018, you were limited to have
    # value1, value2, and value3).
    ifttt_default_title_key = 'value1'

    # The default IFTTT Key to use when mapping the body text to the IFTTT
    # event. The idea here is if someone wants to over-ride the default and
    # change it to another Ingredient Name (in 2018, you were limited to have
    # value1, value2, and value3).
    ifttt_default_body_key = 'value2'

    # The default IFTTT Key to use when mapping the body text to the IFTTT
    # event. The idea here is if someone wants to over-ride the default and
    # change it to another Ingredient Name (in 2018, you were limited to have
    # value1, value2, and value3).
    ifttt_default_type_key = 'value3'

    # IFTTT uses the http protocol with JSON requests
    notify_url = 'https://maker.ifttt.com/' \
                 'trigger/{event}/with/key/{webhook_id}'

    # Define object templates
    templates = (
        '{schema}://{webhook_id}/{events}',
    )

    # Define our template tokens
    template_tokens = dict(NotifyBase.template_tokens, **{
        'webhook_id': {
            'name': _('Webhook ID'),
            'type': 'string',
            'private': True,
            'required': True,
        },
        'events': {
            'name': _('Events'),
            'type': 'list:string',
            'required': True,
        },
    })

    # Define our template arguments
    template_args = dict(NotifyBase.template_args, **{
        'to': {
            'alias_of': 'events',
        },
    })

    # Define our token control
    template_kwargs = {
        'add_tokens': {
            'name': _('Add Tokens'),
            'prefix': '+',
        },
        'del_tokens': {
            'name': _('Remove Tokens'),
            'prefix': '-',
        },
    }

    def __init__(self, webhook_id, events, add_tokens=None, del_tokens=None,
                 **kwargs):
        """
        Initialize IFTTT Object

        add_tokens can optionally be a dictionary of key/value pairs
        that you want to include in the IFTTT post to the server.

        del_tokens can optionally be a list/tuple/set of tokens
        that you want to eliminate from the IFTTT post.  There isn't
        much real functionality to this one unless you want to remove
        reference to Value1, Value2, and/or Value3

        """
        super(NotifyIFTTT, self).__init__(**kwargs)

        # Webhook ID (associated with project)
        self.webhook_id = validate_regex(webhook_id)
        if not self.webhook_id:
            msg = 'An invalid IFTTT Webhook ID ' \
                  '({}) was specified.'.format(webhook_id)
            self.logger.warning(msg)
            raise TypeError(msg)

        # Store our Events we wish to trigger
        self.events = parse_list(events)
        if not self.events:
            msg = 'You must specify at least one event you wish to trigger on.'
            self.logger.warning(msg)
            raise TypeError(msg)

        # Tokens to include in post
        self.add_tokens = {}
        if add_tokens:
            self.add_tokens.update(add_tokens)

        # Tokens to remove
        self.del_tokens = []
        if del_tokens is not None:
            if isinstance(del_tokens, (list, tuple, set)):
                self.del_tokens = del_tokens
```

```python

            elif isinstance(del_tokens, dict):
                # Convert the dictionary into a list
                self.del_tokens = set(del_tokens.keys())

            else:
                msg = 'del_token must be a list; {} was provided'.format(
                    str(type(del_tokens)))
                self.logger.warning(msg)
                raise TypeError(msg)

    def send(self, body, title='', notify_type=NotifyType.INFO, **kwargs):
        """
        Perform IFTTT Notification
        """

        headers = {
            'User-Agent': self.app_id,
            'Content-Type': 'application/json',
        }

        # prepare JSON Object
        payload = {
            self.ifttt_default_title_key: title,
            self.ifttt_default_body_key: body,
            self.ifttt_default_type_key: notify_type,
        }

        # Add any new tokens expected (this can also potentially override
        # any entries defined above)
        payload.update(self.add_tokens)

        # Eliminate fields flagged for removal otherwise ensure all tokens are
        # lowercase since that is what the IFTTT server expects from us.
        payload = {x.lower(): y for x, y in payload.items()
                   if x not in self.del_tokens}

        # error tracking (used for function return)
        has_error = False

        # Create a copy of our event lit
        events = list(self.events)

        while len(events):

            # Retrive an entry off of our event list
            event = events.pop(0)

            # URL to transmit content via
            url = self.notify_url.format(
                webhook_id=self.webhook_id,
                event=event,
            )

            self.logger.debug('IFTTT POST URL: %s (cert_verify=%r)' % (
                url, self.verify_certificate,
            ))
            self.logger.debug('IFTTT Payload: %s' % str(payload))

            # Always call throttle before any remote server i/o is made
            self.throttle()

            try:
                r = requests.post(
                    url,
                    data=dumps(payload),
                    headers=headers,
                    verify=self.verify_certificate,
                    timeout=self.request_timeout,
                )
                self.logger.debug(
                    u"IFTTT HTTP response headers: %r" % r.headers)
                self.logger.debug(
                    u"IFTTT HTTP response body: %r" % r.content)

                if r.status_code != requests.codes.ok:
                    # We had a problem
                    status_str = \
                        NotifyIFTTT.http_response_code_lookup(r.status_code)

                    self.logger.warning(
                        'Failed to send IFTTT notification to {}: '
                        '{}{}error={}.'.format(
                            event,
                            status_str,
                            ', ' if status_str else '',
                            r.status_code))

                    self.logger.debug(
                        'Response Details:\r\n{}'.format(r.content))

                    # Mark our failure
                    has_error = True
                    continue

                else:
                    self.logger.info(
                        'Sent IFTTT notification to %s.' % event)

            except requests.RequestException as e:
                self.logger.warning(
                    'A Connection error occurred sending IFTTT:%s ' % (
                        event) + 'notification.'
                )
                self.logger.debug('Socket Exception: %s' % str(e))

                # Mark our failure
                has_error = True
                continue

        return not has_error

    def url(self, privacy=False, *args, **kwargs):
        """
        Returns the URL built dynamically based on specified arguments.
        """

        # Our URL parameters
        params = self.url_parameters(privacy=privacy, *args, **kwargs)

        # Store any new key/value pairs added to our list
        params.update({'+{}'.format(k): v for k, v in self.add_tokens})
        params.update({'-{}'.format(k): '' for k in self.del_tokens})

        return '{schema}://{webhook_id}@{events}/?{params}'.format(
            schema=self.secure_protocol,
            webhook_id=self.pprint(self.webhook_id, privacy, safe=''),
            events='/'.join([NotifyIFTTT.quote(x, safe='')
                             for x in self.events]),
            params=NotifyIFTTT.urlencode(params),
        )

    @staticmethod
    def parse_url(url):
        """
        Parses the URL and returns enough arguments that can allow
        us to re-instantiate this object.

        """
        results = NotifyBase.parse_url(url, verify_host=False)
        if not results:
            # We're done early as we couldn't load the results
            return results

        # Our API Key is the hostname if no user is specified
        results['webhook_id'] = \
            results['user'] if results['user'] else results['host']

        # Unquote our API Key
        results['webhook_id'] = NotifyIFTTT.unquote(results['webhook_id'])

        # Parse our add_token and del_token arguments (if specified)
        results['add_token'] = results['qsd+']
        results['del_token'] = results['qsd-']

        # Our Event
        results['events'] = list()
        if results['user']:
            # If a user was defined, then the hostname is actually a event
            # too
            results['events'].append(NotifyIFTTT.unquote(results['host']))

        # Now fetch the remaining tokens
        results['events'].extend(NotifyIFTTT.split_path(results['fullpath']))

        # The 'to' makes it easier to use yaml configuration
        if 'to' in results['qsd'] and len(results['qsd']['to']):
            results['events'] += \
                NotifyIFTTT.parse_list(results['qsd']['to'])

        return results

    @staticmethod
    def parse_native_url(url):
        """
        Support https://maker.ifttt.com/use/WEBHOOK_ID/EVENT_ID
        """

        result = re.match(
            r'^https?://maker\.ifttt\.com/use/'
            r'(?P<webhook_id>[A-Z0-9_-]+)'
            r'/?(?P<events>([A-Z0-9_-]+/?)+)?'
            r'/?(?P<params>\?.+)?$', url, re.I)

        if result:
            return NotifyIFTTT.parse_url(
                '{schema}://{webhook_id}{events}{params}'.format(
                    schema=NotifyIFTTT.secure_protocol,
                    webhook_id=result.group('webhook_id'),
                    events='' if not result.group('events')
                    else '@{}'.format(result.group('events')),
                    params='' if not result.group('params')
                    else result.group('params')))

        return None
```