Pacifica Notifications Documentation

David Brown

Apr 02, 2020

Contents:

This service is a Pacifica Policy based routing mechanism for data subscribers to execute workflows based on the availability of data in Pacifica.

CHAPTER 1

Installation

The Pacifica software is available through PyPi so creating a virtual environment to install is what is shown below. Please keep in mind compatibility with the Pacifica Core services.

1.1 Installation in Virtual Environment

These installation instructions are intended to work on both Windows, Linux, and Mac platforms. Please keep that in mind when following the instructions.

Please install the appropriate tested version of Python for maximum chance of success.

1.1.1 Linux and Mac Installation

```
mkdir ~/.virtualenvs
python -m virtualenv ~/.virtualenvs/pacifica
. ~/.virtualenvs/pacifica/bin/activate
pip install pacifica-notifications
```

1.1.2 Windows Installation

This is done using PowerShell. Please do not use Batch Command.

```
mkdir "$Env:LOCALAPPDATA\virtualenvs"
python.exe -m virtualenv "$Env:LOCALAPPDATA\virtualenvs\pacifica"
& "$Env:LOCALAPPDATA\virtualenvs\pacifica\Scripts\activate.ps1"
pip install pacifica-notifications
```

CHAPTER 2

Configuration

The Pacifica Core services require two configuration files. The REST API utilizes CherryPy and review of their configuration documentation is recommended. The service configuration file is a INI formatted file containing configuration for database connections.

2.1 CherryPy Configuration File

An example of Notifications server CherryPy configuration:

```
[global]
log.screen: True
log.access_file: 'access.log'
log.error_file: 'error.log'
server.socket_host: '0.0.0.0'
server.socket_port: 8070

[/]
request.dispatch: cherrypy.dispatch.MethodDispatcher()
tools.response_headers.on: True
tools.response_headers.headers: [('Content-Type', 'application/json')]
```

2.2 Service Configuration File

The service configuration is an INI file and an example is as follows:

```
[notifications]
; This section describes notification specific configurations

; The policy server endpoint to query
policy_url = http://127.0.0.1:8181
```

(continues on next page)

(continued from previous page)

```
[celery]
; This section contains celery messaging configuration

; The broker url is how messages get passed around
broker_url = pyamqp://

; The backend url is how return results are sent around
backend_url = rpc://

[database]
; This section contains database connection configuration

; peewee_url is defined as the URL PeeWee can consume.
; http://docs.peewee-orm.com/en/latest/peewee/database.html#connecting-using-a-
↪database-url
peewee_url = sqliteext:///db.sqlite3

; connect_attempts are the number of times the service will attempt to
; connect to the database if unavailable.
connect_attempts = 10

; connect_wait are the number of seconds the service will wait between
; connection attempts until a successful connection to the database.
connect_wait = 20
```

2.3 Starting the Service

Starting the Notifications service can be done by two methods. However, understanding the requirements and how they apply to REST services is important to address as well. Using the internal CherryPy server to start the service is recommended for Windows platforms. For Linux/Mac platforms it is recommended to deploy the service with uWSGI.

2.3.1 Deployment Considerations

The Notifications service is relatively new and has not seen usage enough to know how it performs.

2.3.2 CherryPy Server

To make running the Notifications service using the CherryPy's builtin server easier we have a command line entry point.

```
$ pacifica-notifications --help
usage: pacifica-notifications [-h] [-c CONFIG] [-p PORT] [-a ADDRESS]

Run the notifications server.

optional arguments:
  -h, --help            show this help message and exit
  -c CONFIG, --config CONFIG
                        cherrypy config file
```

(continues on next page)

(continued from previous page)

```
  -p PORT, --port PORT  port to listen on
  -a ADDRESS, --address ADDRESS
                        address to listen on
$ pacifica-notifications-cmd dbsync
$ pacifica-notifications
[09/Jan/2019:09:17:26] ENGINE Listening for SIGTERM.
[09/Jan/2019:09:17:26] ENGINE Bus STARTING
[09/Jan/2019:09:17:26] ENGINE Set handler for console events.
[09/Jan/2019:09:17:26] ENGINE Started monitor thread 'Autoreloader'.
[09/Jan/2019:09:17:26] ENGINE Serving on http://0.0.0.0:8070
[09/Jan/2019:09:17:26] ENGINE Bus STARTED
```

2.3.3 uWSGI Server

To make running the Notifications service using uWSGI easier we have a module to be included as part of the uWSGI configuration. uWSGI is very configurable and can use this module many different ways. Please consult the uWSGI Configuration documentation for more complicated deployments.

```
$ pip install uwsgi
$ uwsgi --http-socket :8070 --master --module pacifica.notifications.wsgi
```

CHAPTER 3

Example Usage

The (Pacifica Metadata)[https://github.com/pacifica/pacifica-metadata.git] service emits (CloudEvents)[https://github.com/cloudevents/spec] when new data is accepted. This service is intended to recieve and route those events to users that are allowed based on Pacifica Policy.

3.1 API Reference

There are two REST APIs available on this service. The first api accepts cloud events for processing. The second api allows users to subscribe to events and specify routing target urls to send those events.

3.1.1 Cloud Events Recieve

```
POST /receive
Content-Type: application/json
... JSON Cloud Event ...
```

3.1.2 Subscriptions

The subscriptions API is a REST style API accessed on `/eventmatch`.

Create Event Subscription

Request:

```
POST /eventmatch
Http-Remote-User: dmlb2001
Content-Type: application/json
{
```

(continues on next page)

(continued from previous page)

```
  "name": "My Event Match",
  "jsonpath": "data",
  "target_url": "http://www.example.com/recieve"
}
```

Response:

```
Content-Type: application/json
{
  "user": "dmlb2001",
  "updated": "2018-08-02T13:53:05.838827",
  "uuid": "466725b0-cbe1-45cd-b034-c3209aa4b6e0",
  "deleted": null,
  "version": "v0.1",
  "jsonpath": "data",
  "disabled": null,
  "created": "2018-08-02T13:53:05.838827",
  "name": "My Event Match",
  "extensions": {},
  "auth": {},
  "target_url": "http://www.example.com/receive",
  "error": null
}
```

Create Event Subscription with Authentication

Request:

```
POST /eventmatch
Http-Remote-User: dmlb2001
Content-Type: application/json
{
  "name": "My Event Match",
  "jsonpath": "data",
  "auth": {
    "type": "basic",
    "basic": {
      "username": "myusername",
      "password": "password"
    }
  },
  "target_url": "http://www.example.com/recieve"
}
```

Response:

```
Content-Type: application/json
{
  "user": "dmlb2001",
  "updated": "2018-08-02T13:53:05.838827",
  "uuid": "466725b0-cbe1-45cd-b034-c3209aa4b6e0",
  "deleted": null,
  "version": "v0.1",
  "jsonpath": "data",
  "disabled": null,
```

(continues on next page)

(continued from previous page)

```
  "created": "2018-08-02T13:53:05.838827",
  "name": "My Event Match",
  "extensions": {},
  "auth": {
    "type": "basic",
    "basic": {
      "username": "myusername",
      "password": "password"
    }
  },
  "target_url": "http://www.example.com/receive",
  "error": null
}
```

Get Event Subscription

Request:

```
GET /eventmatch/466725b0-cbe1-45cd-b034-c3209aa4b6e0
Http-Remote-User: dmlb2001
Content-Type: application/json
```

Response:

```
Content-Type: application/json
{
  "user": "dmlb2001",
  "updated": "2018-08-02T13:53:05.838827",
  "uuid": "466725b0-cbe1-45cd-b034-c3209aa4b6e0",
  "deleted": null,
  "version": "v0.1",
  "jsonpath": "data",
  "disabled": null,
  "created": "2018-08-02T13:53:05.838827",
  "name": "My Event Match",
  "extensions": {},
  "auth": {},
  "target_url": "http://www.example.com/receive",
  "error": null
}
```

Update Event Subscription

Request:

```
PUT /eventmatch/466725b0-cbe1-45cd-b034-c3209aa4b6e0
Http-Remote-User: dmlb2001
Content-Type: application/json
{
  "target_url": "http://api.example.com/receive"
}
```

Response:

```
Content-Type: application/json
{
  "user": "dmlb2001",
  "updated": "2018-08-02T13:53:05.838827",
  "uuid": "466725b0-cbe1-45cd-b034-c3209aa4b6e0",
  "deleted": null,
  "version": "v0.1",
  "jsonpath": "data",
  "disabled": null,
  "created": "2018-08-02T13:53:05.838827",
  "name": "My Event Match",
  "extensions": {},
  "auth": {},
  "target_url": "http://api.example.com/receive",
  "error": null
}
```

Delete Event Subscription

Request:

```
DELETE /eventmatch/466725b0-cbe1-45cd-b034-c3209aa4b6e0
```

Response:

```
HTTP/1.1 200 OK
```

3.2 Command Line Usage

There are several command line options to manage the database schema and the data within. There are two commands to verify the schema is updated and to update the schema if necessary. There are also commands to manage the growing events that are tracked there.

3.2.1 Schema Management Commands

The database schema can be verified using the `dbchk` subcommand.

```
$ pacifica-notifications-cmd dbchk; echo $?
0 or -1
```

If the result is `-1` then the database is not updated and should be updated. To perform this run the `dbsync` subcommand to update the schema.

```
$ pacifica-notifications-cmd dbsync; echo $?
0
```

The database schema is now updated and the service can now run.

3.2.2 Event Log Management

Events and what matches were calculated are logged in the database and need to be purged on a regular basis depending on the volume of events received. Events can also be queried to view what matches were calculated at the time they were received.

```
$ pacifica-notifications-cmd eventget
Event - a83ead91-8bff-4819-a0a1-d7dcf430b817
{...Some event data...}
    ELM 1791be8f-d9cc-418c-b976-b2a747b58678 (2020-02-29T20:44:17.923416) policy 201␣
↪target 200
    ELM ca028c49-5d87-4ee9-8c2b-620e7c32a10c (2020-02-30T05:13:40.123416) policy 201␣
↪target 200
```

To purge events from the log the `eventpurge` subcommand is used.

```
$ pacifica-notifications-cmd eventpurge --older-than-date '2020-01-01 00:00:00' --
↪limit 1000; echo $?
0
```

CHAPTER 4

Consumer Expectations

4.1 Implementation

4.1.1 How do I develop a consumer?

1. Optionally, develop a shell script that will initialize the environment for the consumer (i.e., that will install any requirements that cannot be listed in a Python3 requirements file).

- The filename of the shell script is `./init.sh`.
- If the shell script file is not present, then the system assumes that it is empty.

1. Optionally, develop a Python3 requirements file for the consumer.

- The filename for the requirements file is `./requirements.txt`.
- If the requirements file is not present, then the system assumes that it is empty.

1. Develop a JSONPath to be tested against the JSON object for a given CloudEvents notification (corresponding to a given Pacifica transaction).

- The filename for the JSONPath is `jsonpath2.txt`.
- The usage of the JSONPath is as follows:
  - If the test returns a non-empty result set, then the consumer will be notified.
  - If the test returns an empty result set, then the consumer will not be notified.

1. Develop a top-level Python3 script (viz., the entry-point) that will be executed by the consumer within a virtual environment. The behavior of the entry-point is to act upon a given CloudEvents notification for a given Pacifica transaction (and its associated input data files and key-value pairs) and then to generate a new Pacifica transaction (and its associated output data files and key-value pairs).

- The filename of the entry-point is `./__main__.py`.
- The usage of the entry-point is `./__main__.py SRC DST`, where:
  - `SRC` is the path to the temporary directory for the incoming Pacifica transaction.

  - `DST` is the path to the temporary directory for the outgoing Pacifica transaction.
- The input data files (downloaded from Pacifica) are located in the `SRC/downloads/` subdirectory.
- The output data files (uploaded to Pacifica) are located in the `DST/uploads/` subdirectory.
- The JSON object for the CloudEvents notification is `SRC/notification.json`.
- The execution of the entry-point terminates with the following exit status codes:
  - `0` = Terminated successfully.
  - `>0` = Terminated unsuccessfully.

1. Compress the entry-point, the requirements file, the JSONPath file and any additional files that are necessary for execution of the entry-point into a zip archive called `consumer.zip`.

Example implementation of a consumer

In this example, we develop a consumer that copies the input data files with all cased characters converted to uppercase.

consumer.zip

The directory-tree listing for `consumer.zip` is as follows:

- /
  - `init.sh`
  - `jsonpath2.txt`
  - `requirements.txt`
  - `__main__.py`

init.sh

The shell script does nothing.

```
#!/usr/bin/env sh

exit 0
```

jsonpath2.txt

The JSONPath returns the set of IDs for input data files whose MIME type is `text/plain`.

```
$[?(@["eventID"] and (@["eventType"] = "org.pacifica.metadata.ingest") and (@["source
↪"] = "/pacifica/metadata/ingest"))]["data"][*][?((@["destinationTable"] = "Files")␣
↪and (@["mimetype"] = "text/plain"))]["_id"]
```

requirements.txt

The requirements file specifies the JSONPath, Pacifica notification service consumer and Promise packages.

```
jsonpath2
pacifica-notifications-consumer
promises
```

__main__.py

The entry-point is as follows:

```
#!/usr/bin/env python3

import json
import os
import shutil
import sys

from jsonpath2 import Path
from pacifica_notifications_consumer import download, upload
from promise import Promise

# execute only if run as a top-level script
if __name__ == '__main__':
  # path to directory for input CloudEvents notification
  orig_event_path = sys.argv[1]

  # path to directory for output CloudEvents notification
  new_event_path = sys.argv[2]

  def upload_did_fulfill(d):
    """Callback for Pacifica uploader promise's eventual value.

    Args:
        d (dict): CloudEvents notification.

    Returns:
        Promise[bool]: True for success, False otherwise.

    Raises:
        BaseException: If an error occurs.
    """

    # delete entire directory tree for input CloudEvents notification
    shutil.rmtree(orig_event_path, ignore_errors=True)

    # delete entire directory tree for output CloudEvents notification
    shutil.rmtree(new_event_path, ignore_errors=True)

    # return True for success
    return Promise(lambda resolve, reject: resolve(True))

  def download_did_fulfill(d):
    """Callback for Pacifica downloader promise's eventual value.
```

(continues on next page)

(continued from previous page)

```
    Args:
        d (dict): CloudEvents notification.

    Returns:
        Promise[bool]: True for success, False otherwise.

    Raises:
        BaseException: If an error occurs.
    """

    # iterate over plain-text files
    for file_d in [ match_data.current_value for match_data in Path.parse_str('$["data
↪"][*][?(@["destinationTable"] = "Files" and @["mimetype"] = "text/plain")]').
↪match(d) ]:
      # open input data file
      with open(os.path.join(orig_event_path, 'downloads', file_d['subdir'], file_d[
↪'name']), 'r') as orig_file:
        # open output data file
        with open(os.path.join(new_event_path, 'uploads', file_d['subdir'], file_d[
↪'name']), 'w') as new_file:
          # read input data file, convert cased characters to uppercase, and then␣
↪write output data file
          new_file.write(orig_file.read().upper())

    # invoke Pacifica uploader with specified transaction attributes and key-value␣
↪pairs
    return upload(new_event_path, {
      'Transactions.instrument': [ match_data.current_value for match_data in Path.
↪parse_str('$["data"][*][?(@["destinationTable"] = "Transactions.instrument")]["value
↪"]').match(d) ][0],
      'Transactions.proposal': [ match_data.current_value for match_data in Path.
↪parse_str('$["data"][*][?(@["destinationTable"] = "Transactions.proposal")]["value"]
↪').match(d) ][0],
      'Transactions.submitter': [ match_data.current_value for match_data in Path.
↪parse_str('$["data"][*][?(@["destinationTable"] = "Transactions.submitter")]["value
↪"]').match(d) ][0],
    }, {
      'Transactions._id': [ match_data.current_value for match_data in Path.parse_str(
↪'$["data"][*][?(@["destinationTable"] = "Transactions._id")]["value"]').match(d)␣
↪][0],
    }).then(upload_did_fulfill)

  # invoke Pacifica downloader
  download(orig_event_path).then(download_did_fulfill)
```

4.1.2 How do I deploy a consumer

1. Download and install the `pacifica-notifications-consumer` package.

- When the `pacifica-notifications-consumer` is successfully installed, the `start-pacifica-notifications-consumer` command is available on the `PATH`.
- When started, the behavior of the `start-pacifica-notifications-consumer` command is to: (i) extract the contents of `consumer.zip` to a temporary location, (ii) create a new virtual environment, (iii) install the contents of the requirements file within said virtual environment, (iv) start a new asynchronous job processing queue, (v) start a new web service end-point for a new consumer, and (vi) register said web service

end-point for said consumer with the given producer.

- When stopped, the behavior of the `start-pacifica-notifications-consumer` command is reverse the side-effects of the start-up behavior (i.e., to clean up after itself).

1. Execute the `start-pacifica-notifications-consumer` command, specifying as command-line arguments:

- The location of `consumer.zip`;
- The URL and authentication credentials for the producer; and
- The configuration for the asynchronous job processing queue, web service end-point, etc.

4.2 Frequently Asked Questions

4.2.1 Are downloaded files persisted after the entry-point for a consumer has terminated?

Yes. Downloaded files may be deleted by the entry-point (e.g., using the `shutil.rmtree` method; see `__main__.py` in the example).

4.2.2 Are locally-generated files persisted after the entry-point for a consumer has terminated?

Yes. Locally-generated files may be deleted by the entry-point (e.g., using the `shutil.rmtree` method; see `__main__.py` in the example).

4.2.3 Can I develop the entry-point for a consumer to wait for two-or-more CloudEvents notifications?

Yes. However, this behavior must be implemented by the entry-points themselves and is not provided by the default behavior of the `start-pacifica-notifications-consumer` command.

For example, to wait for two CloudEvents notifications:

- Develop two consumers with two entry-points.
- The entry-point for the first consumer receives and stores the first CloudEvents notification.
- The entry-point for the second consumer receives the second CloudEvents notification, retrieves the first CloudEvents notification, and then does its work.

4.2.4 Can I develop the entry-point for a consumer using a non-Python3 programming language?

No. Only Python3 is supported as the programming language for the entry-point file (viz., `__main__.py`).

Non-Python3 executables can be called from within the entry-point using the `subprocess` module.

Other programming languages can be called from within the entry-point using the appropriate interface, e.g., the `jpy` package for the Java programming language, and the `rpy2` package for the R programming language.

4.2.5 How do I authenticate with the CloudEvents notification provider?

Authentication credentials are specified in the configuration for the `pacifica-notifications` package (see https://pacifica-notifications.readthedocs.io/en/latest/configuration.html for more information).

Authentication credentials are included in all HTTP requests that are issued by the consumer, e.g., the username is specified via the `Http-Remote-User` header (see https://pacifica-notifications.readthedocs.io/en/latest/exampleusage.html for more information).

4.3 Glossary of Terms

- **Consumer:** A software system that receives CloudEvents notifications (corresponding to Pacifica transactions) from producers. CloudEvents notifications are filtered by testing against a JSONPath. If the test for a given CloudEvents notification is successful, then the consumer routes said CloudEvents notification to a processor.
- **Processor:** A software system that downloads the input data files and metadata for a given Pacifica transaction, processes said input data files and associated metadata, generates output data files and associated metadata, and then creates a new Pacifica transaction.
- **Producer:** A software system that sends CloudEvents notifications (corresponding to Pacifica transactions) to consumers.

CHAPTER 5

Notifications Python Module

5.1 Configuration Python Module

Configuration reading and validation module.

pacifica.notifications.config.**get_config**()
Return the ConfigParser object with defaults set.

5.2 Globals Python Module

Global configuration options expressed in environment variables.

5.3 ORM Python Module

The ORM module defining the SQL model for notifications.

class pacifica.notifications.orm.**BaseModel**(**args*, ***kwargs*)
Auto-generated by pwiz.

DoesNotExist
alias of BaseModelDoesNotExist

_meta = <peewee.Metadata object>

_schema = <peewee.SchemaManager object>

classmethod atomic()
Do the database atomic action.

classmethod database_close()
Close the database connection.

Closing already closed database is not a problem, so continue on.

classmethod database_connect()
Make sure database is connected.

Trying to connect a second time *does* cause problems.

id = <AutoField: BaseModel.id>

class pacifica.notifications.orm.**EventLog**(**args*, ***kwargs*)
Events matching via jsonpath per user.

DoesNotExist
alias of EventLogDoesNotExist

_meta = <peewee.Metadata object>

_schema = <peewee.SchemaManager object>

created = <DateTimeField: EventLog.created>

event_matches

jsondata = <TextField: EventLog.jsondata>

uuid = <UUIDField: EventLog.uuid>

class pacifica.notifications.orm.**EventLogMatch**(**args*, ***kwargs*)
Events matching via jsonpath per user.

DoesNotExist
alias of EventLogMatchDoesNotExist

_meta = <peewee.Metadata object>

_schema = <peewee.SchemaManager object>

created = <DateTimeField: EventLogMatch.created>

event_log = <ForeignKeyField: EventLogMatch.event_log>

event_log_id = <ForeignKeyField: EventLogMatch.event_log>

event_match = <ForeignKeyField: EventLogMatch.event_match>

event_match_id = <ForeignKeyField: EventLogMatch.event_match>

policy_resp_body = <TextField: EventLogMatch.policy_resp_body>

policy_status_code = <TextField: EventLogMatch.policy_status_code>

target_resp_body = <TextField: EventLogMatch.target_resp_body>

target_status_code = <TextField: EventLogMatch.target_status_code>

uuid = <UUIDField: EventLogMatch.uuid>

class pacifica.notifications.orm.**EventMatch**(**args*, ***kwargs*)
Events matching via jsonpath per user.

DoesNotExist
alias of EventMatchDoesNotExist

_meta = <peewee.Metadata object>

_schema = <peewee.SchemaManager object>

auth = <TextField: EventMatch.auth>

created = <DateTimeField: EventMatch.created>

deleted = <DateTimeField: EventMatch.deleted>

disabled = <DateTimeField: EventMatch.disabled>

error = <TextField: EventMatch.error>

event_logs

extensions = <TextField: EventMatch.extensions>

jsonpath = <TextField: EventMatch.jsonpath>

name = <CharField: EventMatch.name>

target_url = <TextField: EventMatch.target_url>

to_hash()
Convert the object to a json serializable hash.

updated = <DateTimeField: EventMatch.updated>

user = <CharField: EventMatch.user>

uuid = <UUIDField: EventMatch.uuid>

validate_jsonpath()
Validate the jsonpath string.

version = <CharField: EventMatch.version>

class pacifica.notifications.orm.**NotificationSystem**(**args*, ***kwargs*)
Notification Schema Version Model.

DoesNotExist
alias of NotificationSystemDoesNotExist

_meta = <peewee.Metadata object>

_schema = <peewee.SchemaManager object>

classmethod get_or_create_version()
Set or create the current version of the schema.

classmethod get_version()
Get the current version as a tuple.

classmethod is_equal()
Check to see if schema version matches code version.

classmethod is_safe()
Check to see if the schema version is safe for the code.

part = <CharField: NotificationSystem.part>

value = <IntegerField: NotificationSystem.value>

class pacifica.notifications.orm.**OrmSync**
Special module for syncing the orm.

This module should incorporate a schema migration strategy.

The supported versions migrating forward must be in a versions array containing tuples for major and minor versions.

The version tuples are directly translated to method names in the orm_update class for the update between those versions.

Example Version Control:

```
class orm_update:
  versions = [
    (0, 1),
    (0, 2),
    (1, 0),
    (1, 1)
  ]

  def update_0_1_to_0_2():
      pass
  def update_0_2_to_1_0():
      pass
```

The body of the update should follow peewee migration practices. http://docs.peewee-orm.com/en/latest/peewee/playhouse.html#migrate

static dbconn_blocking()
Wait for the db connection.

classmethod update_0_0_to_1_0()
Update by adding the new table.

classmethod update_1_0_to_2_0()
Update by adding the auth column.

classmethod update_2_0_to_3_0()
Update by adding the auth column.

classmethod update_tables()
Update the database to the current version.

versions = [(0, 0), (1, 0), (2, 0), (3, 0)]

pacifica.notifications.orm.**eventget**(*args*)
Get events based on command line argument in args.

pacifica.notifications.orm.**eventpurge**(*args*)
Purge events based on command line argument in args.

5.4 REST Python Module

CherryPy module containing classes for rest interface.

class pacifica.notifications.rest.**EventMatch**
CherryPy EventMatch endpoint.

classmethod DELETE(*event_uuid*)
Delete the event by uuid.

classmethod GET(*event_uuid=None*)
Get the event ID and return it.

classmethod POST()
Create an Event Match obj in the database.

classmethod PUT(*event_uuid*)
Update an Event Match obj in the database.

static _http_get(*event_uuid*)
Internal get event by UUID and return peewee obj.

exposed = True

json_schema = {'$ref': '#/definitions/eventmatch', 'definitions': {'eventmatch': {'

class pacifica.notifications.rest.**ReceiveEvent**
CherryPy Receive Event object.

classmethod POST()
Receive the event and dispatch it to backend.

event_json_schema = {}

exposed = True

class pacifica.notifications.rest.**Root**
CherryPy Root Object.

eventmatch = <pacifica.notifications.rest.EventMatch object>

exposed = True

receive = <pacifica.notifications.rest.ReceiveEvent object>

pacifica.notifications.rest.**encode_text**(*thing_obj*)
Encode the text to bytes.

pacifica.notifications.rest.**error_page_default**(***kwargs*)
The default error page should always enforce json.

pacifica.notifications.rest.**get_remote_user**()
Get the remote user from cherrypy request headers.

5.5 JSON Path Python Module

The jsonpath interface module.

pacifica.notifications.jsonpath.**find**(*expr*, *data*)
Match the expression in the data.

pacifica.notifications.jsonpath.**parse**(*jsonpath_str*)
Parse the json path.

5.6 Celery Tasks Python Module

The Celery tasks module.

pacifica.notifications.tasks.**create_log_match**(*eventmatch*, *event_log_uuid*, *policy_resp*)
Create the EventLogMatch object.

pacifica.notifications.tasks.**disable_eventmatch**(*eventmatch_uuid*, *error*)
Disable the eventmatch obj.

pacifica.notifications.tasks.**dispatch_orm_event**(*orm_event*)
Dispatch the event from an existing orm obj.

pacifica.notifications.tasks.**event_auth_to_requests**(*eventmatch*, *headers*)
Convert the eventmatch authentication to requests arguments.

pacifica.notifications.tasks.**update_log_match**(*elm_uuid*, *target_resp*)
Update the EventLogMatch object with the target resp.

5.7 WSGI Python Module

The WSGI interface module for notifications. Pacifica Notifications Module.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

Symbols

A

B

C

D

E

F

G

I

J

N

O

P

R

T

U

V