Pacifica UniqueID Documentation

David Brown

May 18, 2020

Contents:

The Pacifica UniqueID service provides unique auto-incrementing integers to the Pacifica Core services.

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-uniqueid
```

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-uniqueid
```

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 UniqueID 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: 8051

[/]
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:

```
[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
```

(continues on next page)

(continued from previous page)

```
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 UniqueID 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 UniqueID service requirements make it particularly suseptible to caching layers within other systems helping to much. The requirement is that sequential calls to the service should provide unique IDs (in the form of a range). The sequential calls could be called in a tight loop and have the exact same parameters. However, unlike most REST services the return value must be unique.

This presents a problem for placement of this service among the rest of the Pacifica Core services. It is highly recommended that the UniqueID service be deployed with a single worker to handle requests and no caching in front of it.

For large deployments network accessable databases are preferred for many reasons. Keep in mind that distributing UniqueID services among Pacifica services to configure them all with the same `peewee_url`.

The primary consumer of the UniqueID service is the Ingest service. The Ingest service has two parts frontend and backend. Each part of the Ingest service requests different `modes` from the UniqueID service. So, placement of the UniqueID service near the Ingest frontend and backend is recommended for good performance.

2.3.2 CherryPy Server

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

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

Run the uniqueid server.

optional arguments:
  -h, --help            show this help message and exit
  -c CONFIG, --config CONFIG
                        cherrypy config file
  -p PORT, --port PORT  port to listen on
  -a ADDRESS, --address ADDRESS
                        address to listen on
```

(continues on next page)

(continued from previous page)

```
$ pacifica-uniqueid-cmd dbsync
$ pacifica-uniqueid
[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://0.0.0.0:8051
[09/Jan/2019:09:17:26] ENGINE Bus STARTED
```

2.3.3 uWSGI Server

To make running the UniqueID 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 :8051 --master -p 1 --module pacifica.uniqueid.wsgi
```

CHAPTER 3

Example Usage

The usage of this API is through REST endpoint `/getid`.

3.1 The API

Get a range of unique IDs for a given mode.

```
$ curl 'http://localhost:8051/getid?range=10&mode=file'
{"endIndex": 9, "startIndex": 0}
```

Select a different mode to get different unique IDs. If a mode is currently unsupported by the database, a new row will be started to support it.

```
$ curl 'http://localhost:8000/getid?range=10&mode=file'
{"endIndex": 9, "startIndex": 0}
$ curl 'http://localhost:8000/getid?range=10&mode=upload'
{"endIndex": 9, "startIndex": 0}
```

CHAPTER 4

UniqueID Python Module

4.1 Configuration Python Module

Configuration reading and validation module.

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

4.2 Globals Python Module

Global configuration options expressed in environment variables.

4.3 ORM Python Module

ORM for index server.

class pacifica.uniqueid.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 `OrmSync` class for the update between those versions.

Example Version Control:

```
class OrmSync:
    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 an update method 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 boolean column.

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

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

class pacifica.uniqueid.orm.**UniqueIndex**(**args*, ***kwargs*)
Auto-generated by pwiz maps a python record to a mysql table.

DoesNotExist
alias of UniqueIndexDoesNotExist

_meta = <peewee.Metadata object>

_schema = <peewee.SchemaManager object>

classmethod atomic()
Get the atomic context or decorator.

classmethod database_close()
Close the database connection.

Closing already closed database throws an error so catch it and continue on.

classmethod database_connect()
Make sure database is connected.

Trying to connect a second time doesnt cause any problems.

idid = <CharField: UniqueIndex.idid>

index = <BigIntegerField: UniqueIndex.index>

class pacifica.uniqueid.orm.**UniqueIndexBase**(**args*, ***kwargs*)
UniqueIndex base model for database setup.

DoesNotExist
alias of UniqueIndexBaseDoesNotExist

_meta = <peewee.Metadata object>

_schema = <peewee.SchemaManager object>

id = <AutoField: UniqueIndexBase.id>

class pacifica.uniqueid.orm.**UniqueIndexSystem**(**args*, ***kwargs*)
UniqueIndex Schema Version Model.

DoesNotExist
alias of UniqueIndexSystemDoesNotExist

_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: UniqueIndexSystem.part>

value = <IntegerField: UniqueIndexSystem.value>

pacifica.uniqueid.orm.**update_index**(*id_range*, *id_mode*)
Update the index for a mode and returns a unique start and stop index.

4.4 REST Python Module

UniqueID CherryPy Module.

class pacifica.uniqueid.rest.**GetID**
CherryPy GetID object.

static GET(***kwargs*)
Get an id range for the mode.

exposed = True

class pacifica.uniqueid.rest.**Root**
CherryPy Root object.

static GET()
Return happy message about functioning service.

exposed = True

getid = <pacifica.uniqueid.rest.GetID object>

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

4.5 WSGI Python Module

The WSGI interface module for uniqueid. This is the uniqueid module.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

Symbols

A

D

E

G

I

O

P

R

U

V