jaraco.postgres

Release 5.1.dev0+g5737504.d20190101

Jan 01, 2019

Contents

CHAPTER 1

History

1.1 5.0

01 Jan 2019

Switch to pkgutil namespace technique for the `jaraco` namespace.

1.2 4.3

28 Dec 2018

#6: Avoid use of deperecated `yield_fixture`.

1.3 4.2

28 Nov 2018

Refresh package metadata.

1.4 4.1

03 Apr 2018

Added PostgresDatabase.ensure_user, which will not fail if the user already exists. In PostgresServer.create, use ensure_user to allow multiple databases to be created using the same user.

1.5 4.0

02 Apr 2018

PostgresDatabase no longer uses 'pmxtest' as the default user, but instead the username defaults to the database name.

1.6 3.0.1

30 Mar 2018

#2: When searching heuristic paths, sort parsed version numbers numerically so that 10 is greater than 9.

1.7 3.0

17 Oct 2017

Removed global variables and behavior on import:

- root
- INITDB
- PG_CTL
- PSQL
- POSTGRES

Instead, use `PostgresFinder.find_root()`.

1.8 2.0.1

17 Oct 2017

#1: Fix version detection on Postgres 10.0.

1.9 2.0

19 Sep 2017

PostgresServer now creates all databases with a default locale of `en_US.UTF-8`. Now, databases will be Unicode-capable by default. To restore the original behavior, deferring to the system locale, pass `locale=None` to `initdb`.

1.10 1.6

26 Jul 2016

Add `.create` convenience method to PostgresServer.

1.11 1.5

21 Jun 2016

Project is now released automatically through Travis Continuous Integration.

Add Trove classifier for Pytest Framework.

1.12 1.4

26 Dec 2015

Moved hosting to Github and updated skeleton.

1.13 1.3

07 Dec 2015

Add `postgresql_instance` fixture.

1.14 1.2

05 Nov 2015

Added some logging info to initdb, start, stop.

Substantial improvements for test suite.

1.15 1.1

17 Jun 2015

Allow the POSTGRESQL_HOME environment variable to stipulate where the Postgres binaries must be found, overriding heuristic search.

1.16 1.0

05 May 2015

Initial release, using models from yg.test 4.1. This module provides helpers for creating and managing databases.

This technique may be helpful in production code, and it's especially relevant to functional tests which depend on test databases.

Note that this package is more "persistent" than you might expect. The Postgres servers created by this module will remain alive after this module terminates. A future instance of this module, running in a different process, can adopt that postgres server and manage it without a hitch.

Thus this module can create a postgres server at one time, then be reborn in another process at another time and continue to manage that server. It can also manage postgres servers and databases that were created elsewhere.

This capability is useful for production code, where many things may happen between server launch and server shutdown.

The key to this flexibility is that the DBMS can be located by pathname to the storage directory. That pathname handle leads to the server's PID, status, etc.

Warning: These methods are inconsistent about the exceptions that they raise. Some errors provoke OSError, whereas other similar errors might provoke CalledProcessError or RuntimeError. This should be made consistent.

exception jaraco.postgres.**NotInitializedError**
Bases: Exception

An exception raised when an uninitialized DBMS is asked to do something

class jaraco.postgres.**PostgresDatabase**(*db_name*, *user=None*, *host='localhost'*, *port=5432*, *superuser='postgres'*, *template='template1'*)
Bases: object

Typical usage: db = PostgresDatabase(db_name='test_db', user='test_user') db.create_user() db.create('CREATE TABLE foo (value text not null); ...') db.sql('Ad-hoc string of SQL...') ... db.drop() db.drop_user()

create(*sql=None*)
CREATE this DATABASE.

@param sql: (Optional) A string of psql (such as might be generated by pg_dump); it will be executed by psql(1) after creating the database. @type sql: str

@rtype: None

create_user()
CREATE this USER.

Beware that this method is open to SQL injection attack. Don't use unvetted values of self.user.

drop()
DROP this DATABASE, if it exists.

drop_if_exists()
DROP this DATABASE, if it exists.

drop_user()
DROP this USER, if it exists.

ensure_user()
Create the user but only if it does not yet exist.

psql(*args*)
Invoke psql, passing the given command-line arguments.

Typical <args> values: ['-c', <sql_string>] or ['-f', <pathname>].

Connection parameters are taken from self. STDIN, STDOUT, and STDERR are inherited from the parent.

WARNING: This method uses the psql(1) program, which ignores SQL errors by default. That hides many real errors, making our software less reliable. To overcome this flaw, add this line to the head of your SQL:

"set ON_ERROR_STOP TRUE"

@return: None. Raises an exception upon error, but *ignores SQL errors* unless "set ON_ERROR_STOP TRUE" is used.

psql_string(*sql*)
Evaluate the sql file (possibly multiple statements) using psql.

sql(*input_string*, **args*)
Execute a SQL command using the Python DBI directly.

Connection parameters are taken from self. Autocommit is in effect.

Example: .sql('SELECT %s FROM %s WHERE age > %s', 'name', 'table1', '45')

@param input_string: A string of SQL. May contain %s or %(name)s format specifiers; they are replaced with corresponding values taken from args.

@param args: zero or more parameters to interpolate into the string. Note that they're passed individually, not as a single tuple.

@return: Whatever .fetchall() returns.

super_psql(*args*)
Just like .psql(), except that we connect as the database superuser (and we connect to the superuser's database, not the user's database).

class jaraco.postgres.**PostgresFinder**
Bases: jaraco.services.paths.PathFinder

args = ['--version']

candidate_paths = ['', '/usr/local/pgsql/bin/', '/Program Files/pgsql/bin', '/usr/lib/

env_paths = []

exe = 'pg_ctl'

heuristic_paths = ['', '/usr/local/pgsql/bin/', '/Program Files/pgsql/bin', '/usr/lib/

class jaraco.postgres.**PostgresServer**(*host='localhost'*, *port=5432*, *base_pathname=None*, *superuser='postgres'*)
Bases: object

create(*db_name*, ***kwargs*)
Construct a PostgresDatabase and create it on self

data_directory

destroy()
Undo the effects of initdb.

Destroy all evidence of this DBMS, including its backing files.

get_setting(*name*)

static get_version()
Returns the Postgres version in tuple form, e.g: (9, 1)

initdb(*quiet=True*, *locale='en_US.UTF-8'*)
Bootstrap this DBMS from nothing.

If you're running in an environment where the DBMS is provided as part of the basic infrastructure, you probably don't want to call this method!

@param quiet: Should we operate quietly, emitting nothing if things go well?

is_running(*tries=10*)
Return True if this server is currently running and reachable.

The postgres tools have critical windows during which they give misbehave or give the wrong answer. If postgres was just launched:

- it might not yet appear to be running, or

- pg_ctl might think that it's running, but psql might not yet be able to connect to it, or
- it might be about to abort because of a configuration problem,
- or all three! It might be starting up, but about to abort.

Sadly, it's not easy to make a declaration about state if the server just started or stopped. To increase confidence, makes repeated checks, and declares a decision only after <tries> consecutive measurements agree.

pid
The server's PID (None if not running).

ready()
Assumes postgres now talks to pg_ctl, but might not yet be listening or connections from psql. Test that psql is able to connect, as it occasionally takes 5-10 seconds for postgresql to start listening.

start()
Launch this postgres server. If it's already running, do nothing.

If the backing storage directory isn't configured, raise NotInitializedError.

This method is optional. If you're running in an environment where the DBMS is provided as part of the basic infrastructure, you probably want to skip this step!

stop()
Stop this DMBS daemon. If it's not currently running, do nothing.

Don't return until it's terminated.

jaraco.postgres.fixtures.**postgresql_instance**()

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

j

Index