# Momoko Documentation

***Release 1.0.0***

**Frank Smit**

January 28, 2013

# CONTENTS

Wraps (asynchronous) Psycopg2 for Tornado.

Momoko wraps Psycopg2 to make it suitable for use in Tornado using Psycopg2's support for asynchronous connections. Momoko's API is geared towards use with Tornado's gen module, but also works fine without it.

Contents:

CHAPTER

ONE

# CHANGELOG

## 1.1 1.0.0b2 (2013-01-??)

- Add and remove a database connection to and from the IOLoop for each operation. See pull request 38 and commits 189323211b and 92940db0a0 for more information.
- Add support for hstore.

## 1.2 1.0.0b1 (2012-12-16)

This is a beta release. It means that the code has not been tested thoroughly yet. This first beta release is meant to provide all the functionality of the previous version plus a few additions.

- Most of the code has been rewritten.
- The mogrify method has been added.
- Added support for transactions.
- The query chain and batch have been removed, because `tornado.gen` can be used instead.
- Error reporting has bee improved by passing the raised exception to the callback. A callback accepts two arguments: the cursor and the error.
- `Op`, `WaitOp` and `WaitAllOps` in `momoko.utils` are wrappers for classes in `tornado.gen` which raise the error again when one occurs. And the user can capture the exception in the request handler.
- A complete set of tests has been added in the `momoko` module: `momoko.tests`. These can be run with `python setup.py test`.

## 1.3 0.5.0 (2012-07-30)

- Removed all Adisp related code.
- Refactored connection pool and connection polling.
- Just pass all unspecified arguments to `BlockingPool` and `AsyncPool`. So `connection_factory` can be used again.

## 1.4 0.4.0 (2011-12-15)

- Reorganized classes and files.
- Renamed `momoko.Client` to `momoko.AsyncClient`.
- Renamed `momoko.Pool` to `momoko.AsyncPool`.
- Added a client and pool for blocking connections, `momoko.BlockingClient` and `momoko.BlockingPool`.
- Added `PoolError` to the import list in `__init__.py`.
- Added an example that uses Tornado's gen module and Swift.
- Callbacks are now optional for `AsyncClient`.
- `AsyncPool` and `Poller` now accept a ioloop argument. [fzzbt]
- Unit tests have been added. [fzzbt]

## 1.5 0.3.0 (2011-08-07)

- Renamed `momoko.Momoko` to `momoko.Client`.
- Programming in blocking-style is now possible with `AdispClient`.
- Support for Python 3 has been added.
- The batch and chain fucntion now accepts different arguments. See the documentation for details.

## 1.6 0.2.0 (2011-04-30)

- Removed `executemany` from `Momoko`, because it can not be used in asynchronous mode.
- Added a wrapper class, `Momoko`, for `Pool`, `BatchQuery` and `QueryChain`.
- Added the `QueryChain` class for executing a chain of queries (and callables) in a certain order.
- Added the `BatchQuery` class for executing batches of queries at the same time.
- Improved `Pool._clean_pool`. It threw an `IndexError` when more than one connection needed to be closed.

## 1.7 0.1.0 (2011-03-13)

- Initial release.

CHAPTER

TWO

# INSTALLATION

Momoko supports Python 2 and 3 and PyPy with psycopg2ct or psycopg2cffi. And the only dependencies are Tornado and Psycopg2 (or psycopg2ct/psycopg2cffi). Installation is easy using *easy_install* or pip:

```
pip install momoko
```

The lastest source code can always be cloned from the Github repository with:

```
git clone git://github.com/FSX/momoko.git
cd momoko
python setup.py install
```

Psycopg2 is used by default when installing Momoko, but psycopg2ct or psycopg2cffi can also be used by setting the `MOMOKO_PSYCOPG2_IMPL` environment variable to `psycopg2ct` or `psycopg2cffi` before running `setup.py`. For example:

```
# 'psycopg2', 'psycopg2ct' or 'psycopg2cffi'
export MOMOKO_PSYCOPG2_IMPL='psycopg2cffi'
```

The unit tests als use this variable. It needs to be set if something else is used instead of Psycopg2 when running the unit tests. Besides `MOMOKO_PSYCOPG2_IMPL` there are also other variables that need to be set for the unit tests.

Here's an example for the environment variables:

```
export MOMOKO_TEST_DB='your_db'  # Default: momoko_test
export MOMOKO_TEST_USER='your_user'  # Default: postgres
export MOMOKO_TEST_PASSWORD='your_password'  # Empty de default
export MOMOKO_TEST_HOST='localhost'  # Empty de default
export MOMOKO_TEST_PORT='5432'  # Default: 5432

# Set to '0' if hstore extension isn't enabled
export MOMOKO_TEST_HSTORE='1'  # Default: 0
```

And running the tests is easy:

```
python setup.py test
```

CHAPTER

THREE

# TUTORIAL

How does stuff work?

- Settings things up
- Connecting
- One simple query
- Callproc and mogrify
- Using tornado.gen
- Simulating a batch (with a list of gen.Task), WaitOp, WaitAll
- Difference betweens Momoko's utilities and Tornado's gen module.
- Handling errors

CHAPTER

FOUR

# TESTING

Stuff about unit tests.

CHAPTER

FIVE

# API

Classes, methods and stuff.

## 5.1 Connections

**class** momoko.**Pool**(*dsn*, *connection_factory=None*, *register_hstore=False*, *minconn=1*, *maxconn=5*, *cleanup_timeout=10*, *callback=None*, *ioloop=None*)

Asynchronous connection pool.

The pool manages database connections and passes operations to connections.

See momoko.Connection for documentation about the dsn and connection_factory parameters. These are used by the connection pool when a new connection is created.

**Parameters**

- **register_hstore** (*boolean*) – Register adapter and typecaster for dict-hstore conversions.
- **minconn** (*integer*) – Amount of connections created upon initialization. Defaults to 1.
- **maxconn** (*integer*) – Maximum amount of connections allowed by the pool. Defaults to 5.
- **cleanup_timeout** (*integer*) – Time in seconds between pool cleanups. Unused connections are closed and removed from the pool until only minconn are left. When an integer below 1, like -1 is used the pool cleaner will be disabled. Defaults to 10.
- **callback** (*callable*) – A callable that's called after all the connections are created. Defaults to None.
- **ioloop** – An instance of Tornado's IOLoop. Defaults to None.

**callproc**(*procname*, *parameters=()*, *cursor_factory=None*, *callback=None*, *connection=None*)

Call a stored database procedure with the given name.

See momoko.Connection.callproc() for documentation about the parameters. The connection parameter is for internal use.

**close**()

Close the connection pool.

**execute**(*operation*, *parameters=()*, *cursor_factory=None*, *callback=None*, *connection=None*)

Prepare and execute a database operation (query or command).

See momoko.Connection.execute() for documentation about the parameters. The connection parameter is for internal use.

**mogrify**(*operation*, *parameters=()*, *callback=None*, *connection=None*)
Return a query string after arguments binding.

See `momoko.Connection.mogrify()` for documentation about the parameters. The `connection` parameter is for internal use.

**new**(*callback=None*)
Create a new connection and add it to the pool.

**Parameters** **callback** (*callable*) – A callable that's called after the connection is created. It accepts one paramater: an instance of `momoko.Connection`. Defaults to `None`.

**transaction**(*statements*, *cursor_factory=None*, *callback=None*, *connection=None*)
Run a sequence of SQL queries in a database transaction.

See `momoko.Connection.transaction()` for documentation about the parameters. The `connection` parameter is for internal use.

**class** `momoko.`**Connection**(*dsn*, *connection_factory=None*, *callback=None*, *ioloop=None*)
Create an asynchronous connection.

**Parameters**

- **dsn** (*string*) – A Data Source Name string containing one of the following values:
  - **dbname** - the database name
  - **user** - user name used to authenticate
  - **password** - password used to authenticate
  - **host** - database host address (defaults to UNIX socket if not provided)
  - **port** - connection port number (defaults to 5432 if not provided)

  Or any other parameter supported by PostgreSQL. See the PostgreSQL documentation for a complete list of supported parameters.
- **connection_factory** – The `connection_factory` argument can be used to create non-standard connections. The class returned should be a subclass of psycopg2.extensions.connection. See Connection and cursor factories for details. Defaults to `None`.
- **callback** (*callable*) – A callable that's called after the connection is created. It accepts one paramater: an instance of `momoko.Connection`. Defaults to `None`.
- **ioloop** – An instance of Tornado's IOLoop. Defaults to `None`.

**busy**()
Check if the connection is busy or not.

**callproc**(*procname*, *parameters=()*, *cursor_factory=None*, *callback=None*)
Call a stored database procedure with the given name.

The sequence of parameters must contain one entry for each argument that the procedure expects. The result of the call is returned as modified copy of the input sequence. Input parameters are left untouched, output and input/output parameters replaced with possibly new values.

The procedure may also provide a result set as output. This must then be made available through the standard fetch*() methods.

**Parameters**

- **operation** (*string*) – The name of the database procedure.

- **parameters** (*tuple/list*) – A list or tuple with query parameters. See Passing parameters to SQL queries for more information. Defaults to an empty tuple.
- **cursor_factory** – The `cursor_factory` argument can be used to create non-standard cursors. The class returned must be a subclass of psycopg2.extensions.cursor. See Connection and cursor factories for details. Defaults to `None`.
- **callback** (*callable*) – A callable that is executed when the query has finished. It must accept two positional parameters. The first one being the cursor and the second one `None` or an instance of an exception if an error has occurred, in that case the first parameter will be `None`. Defaults to `None`.

**close**()
Remove the connection from the IO loop and close it.

**closed**
Indicates whether the connection is closed or not.

**execute**(*operation*, *parameters=()*, *cursor_factory=None*, *callback=None*)
Prepare and execute a database operation (query or command).

**Parameters**

- **operation** (*string*) – An SQL query.
- **parameters** (*tuple/list*) – A list or tuple with query parameters. See Passing parameters to SQL queries for more information. Defaults to an empty tuple.
- **cursor_factory** – The `cursor_factory` argument can be used to create non-standard cursors. The class returned must be a subclass of psycopg2.extensions.cursor. See Connection and cursor factories for details. Defaults to `None`.
- **callback** (*callable*) – A callable that is executed when the query has finished. It must accept two positional parameters. The first one being the cursor and the second one `None` or an instance of an exception if an error has occurred, in that case the first parameter will be `None`. Defaults to `None`.

**mogrify**(*operation*, *parameters=()*, *callback=None*)
Return a query string after arguments binding.

The string returned is exactly the one that would be sent to the database running the execute() method or similar.

**Parameters**

- **operation** (*string*) – An SQL query.
- **parameters** (*tuple/list*) – A list or tuple with query parameters. See Passing parameters to SQL queries for more information. Defaults to an empty tuple.
- **callback** (*callable*) – A callable that is executed when the query has finished. It must accept two positional parameters. The first one being the resulting query as a byte string and the second one `None` or an instance of an exception if an error has occurred. Defaults to `None`.

**transaction**(*statements*, *cursor_factory=None*, *callback=None*)
Run a sequence of SQL queries in a database transaction.

**Parameters**

- **statements** (*tuple/list*) – List or tuple containing SQL queries with or without parameters. An item can be a string (SQL query without parameters) or a tuple/list with two items, an SQL query and a tuple/list wuth parameters. An example:

See Passing parameters to SQL queries for more information.

- **cursor_factory** – The `cursor_factory` argument can be used to create non-standard cursors. The class returned must be a subclass of psycopg2.extensions.cursor. See Connection and cursor factories for details. Defaults to `None`.
- **callback** (*callable*) – A callable that is executed when the transaction has finished. It must accept two positional parameters. The first one being a list of cursors in the same order as the given statements and the second one `None` or an instance of an exception if an error has occurred, in that case the first parameter is an empty list. Defaults to `None`.

## 5.2 Utilities

**class** momoko.**Op**(*func*, **args*, ***kwargs*)
Run a single asynchronous operation.

Behaves like tornado.gen.Task, but raises an exception (one of Psycop2's exceptions) when an error occurs related to Psycopg2 or PostgreSQL.

**class** momoko.**WaitOp**(*key*)
Return the argument passed to the result of a previous tornado.gen.Callback.

Behaves like tornado.gen.Wait, but raises an exception (one of Psycop2's exceptions) when an error occurs related to Psycopg2 or PostgreSQL.

**class** momoko.**WaitAllOps**(*keys*)
Return the results of multiple previous tornado.gen.Callback.

Behaves like tornado.gen.WaitAll, but raises an exception (one of Psycop2's exceptions) when an error occurs related to Psycopg2 or PostgreSQL.

## 5.3 Exceptions

**class** momoko.**PoolError**
The `PoolError` exception is raised when something goes wrong in the connection pool. When the maximum amount is exceeded for example.

CHAPTER

SIX

# INDICES AND TABLES

- *genindex*
- *search*

# PYTHON MODULE INDEX

# INDEX