Contents:
1.1 Introduction

pymonetdb is a native python client API for monetDB. This API is cross-platform, and doesn’t depend on any monetdb libraries. It has support for python 3.5+ and PyPy and is Python DBAPI 2.0 compatible.

Note: Since June 2016 pymonetdb is now the official MonetDB Python API. It replaces the old python-monetdb code. pymonetdb should be a drop-in replacement for python-monetdb. The only thing that changes is the module name; change import monetdb into import pymonetdb.

1.2 Installation

To install the MonetDB python API run the following command from the python source directory:

```
$ python setup.py install
```

pymonetdb is also available on pypi:

```
$ pip install pymonetdb
```

That’s all, now you are ready to start using the API.
Examples usage below:

```python
> # import the SQL module
> import pymonetdb
> # set up a connection. arguments below are the defaults
> connection = pymonetdb.connect(username="monetdb", password="monetdb",
    hostname="localhost", database="demo")
> # create a cursor
> cursor = connection.cursor()
> # increase the rows fetched to increase performance (optional)
> cursor.arraysize = 100
> # execute a query (return the number of rows to fetch)
> cursor.execute('SELECT * FROM tables')
26
> # fetch only one row
> cursor.fetchone()
[1062, 'schemas', 1061, None, 0, True, 0, 0]
> # fetch the remaining rows
> cursor.fetchall()
[[1067, 'types', 1061, None, 0, True, 0, 0],
 [1076, 'functions', 1061, None, 0, True, 0, 0],
 [1085, 'args', 1061, None, 0, True, 0, 0],
 [1093, 'sequences', 1061, None, 0, True, 0, 0],
 [1103, 'dependencies', 1061, None, 0, True, 0, 0],
 [1107, 'connections', 1061, None, 0, True, 0, 0],
 [1116, '_tables', 1061, None, 0, True, 0, 0],...
 [4141, 'user_role', 1061, None, 0, True, 0, 0],
(continues on next page)
If you would like to communicate with the database at a lower level you can use the MAPI library:

```python
from pymonetdb import mapi
server = mapi.Connection()
server.connect(hostname="localhost", port=50000, username="monetdb",
               password="monetdb", database="demo", language="sql")
server.cmd("SELECT * FROM tables;")
...```

```sql
[4144, 'auths', 1061, None, 0, True, 0, 0],
[4148, 'privileges', 1061, None, 0, True, 0, 0]]
> # Show the table meta data
cursor.description
[('id', 'int', 4, 4, None, None, None),
 ('name', 'varchar', 12, 12, None, None, None),
 ('schema_id', 'int', 4, 4, None, None, None),
 ('query', 'varchar', 168, 168, None, None, None),
 ('type', 'smallint', 1, 1, None, None, None),
 ('system', 'boolean', 5, 5, None, None, None),
 ('commit_action', 'smallint', 1, 1, None, None, None),
 ('temporary', 'tinyint', 1, 1, None, None, None)]
```
3.1 Basic SQL usage

```python
class pymonetdb.sql.connections.Connection (database, hostname=None, port=50000,
  username='monetdb', password='monetdb',
  unix_socket=None, autocommit=False,
  host=None, user=None, connect_timeout=-1)
```

Bases: object

A MonetDB SQL database connection

**exception DataError**

Bases: `pymonetdb.exceptions.DatabaseError`

Exception raised for errors that are due to problems with the processed data like division by zero, numeric value out of range, etc. It must be a subclass of `DatabaseError`.

**exception DatabaseError**

Bases: `pymonetdb.exceptions.Error`

Exception raised for errors that are related to the database. It must be a subclass of `Error`.

**exception Error**

Bases: `Exception`

Exception that is the base class of all other error exceptions. You can use this to catch all errors with one single `except` statement. Warnings are not considered errors and thus should not use this class as base. It must be a subclass of the Python `StandardError` (defined in the module exceptions).

**exception IntegrityError**

Bases: `pymonetdb.exceptions.DatabaseError`

Exception raised when the relational integrity of the database is affected, e.g. a foreign key check fails. It must be a subclass of `DatabaseError`.

**exception InterfaceError**

Bases: `pymonetdb.exceptions.Error`
Exception raised for errors that are related to the database interface rather than the database itself. It must be a subclass of Error.

**exception InternalError**
- Bases: `pymonetdb.exceptions.DatabaseError`

Exception raised when the database encounters an internal error, e.g. the cursor is not valid anymore, the transaction is out of sync, etc. It must be a subclass of DatabaseError.

**exception NotSupportedError**
- Bases: `pymonetdb.exceptions.DatabaseError`

Exception raised in case a method or database API was used which is not supported by the database, e.g. requesting a `rollback()` on a connection that does not support transaction or has transactions turned off. It must be a subclass of DatabaseError.

**exception OperationalError**
- Bases: `pymonetdb.exceptions.DatabaseError`

Exception raised for errors that are related to the database’s operation and not necessarily under the control of the programmer, e.g. an unexpected disconnect occurs, the data source name is not found, a transaction could not be processed, a memory allocation error occurred during processing, etc. It must be a subclass of DatabaseError.

**exception ProgrammingError**
- Bases: `pymonetdb.exceptions.DatabaseError`

Exception raised for programming errors, e.g. table not found or already exists, syntax error in the SQL statement, wrong number of parameters specified, etc. It must be a subclass of DatabaseError.

**exception Warning**
- Bases: `Exception`

Exception raised for important warnings like data truncations while inserting, etc. It must be a subclass of the Python StandardError (defined in the module exceptions).

**close()**
- Close the connection.

The connection will be unusable from this point forward; an Error exception will be raised if any operation is attempted with the connection. The same applies to all cursor objects trying to use the connection. Note that closing a connection without committing the changes first will cause an implicit rollback to be performed.

**command**(command)
- use this function to send low level mapi commands

**commit()**
- Commit any pending transaction to the database. Note that if the database supports an auto-commit feature, this must be initially off. An interface method may be provided to turn it back on.

Database modules that do not support transactions should implement this method with void functionality.

**cursor()**
- Return a new Cursor Object using the connection. If the database does not provide a direct cursor concept, the module will have to emulate cursors using other means to the extent needed by this specification.

**default_cursor**
- alias of `pymonetdb.sql.cursors.Cursor`

**execute**(query)
- use this for executing SQL queries
gettimeout()  
get the amount of time before a connection times out

rollback()  
This method is optional since not all databases provide transaction support.

In case a database does provide transactions this method causes the database to roll back to the start of any pending transaction. Closing a connection without committing the changes first will cause an implicit rollback to be performed.

set_autocommit(autocommit)  
Set auto commit on or off. ‘autocommit’ must be a boolean

set_replysize(replysize)  
set_sizeheader(sizeheader)  
Set sizeheader on or off. When enabled monetdb will return the size a type. ‘sizeheader’ must be a boolean.

set_timezone(seconds_east_of_utc)  
settimeout(timeout)  
set the amount of time before a connection times out

class pymonetdb.sql.cursors.Cursor(connection)  
Bases: object

This object represents a database cursor, which is used to manage the context of a fetch operation. Cursors created from the same connection are not isolated, i.e., any changes done to the database by a cursor are immediately visible by the other cursors

close()  
Close the cursor now (rather than whenever __del__ is called). The cursor will be unusable from this point forward; an Error (or subclass) exception will be raised if any operation is attempted with the cursor.

dbg(query, fname, sample=-1)  
Locally debug a given Python UDF function in a SQL query using the PDB debugger. Optionally can run on only a sample of the input data, for faster data export.

execute(operation, parameters=None)  
Prepare and execute a database operation (query or command). Parameters may be provided as mapping and will be bound to variables in the operation.

executemany(operation, seq_of_parameters)  
Prepare a database operation (query or command) and then execute it against all parameter sequences or mappings found in the sequence seq_of_parameters.

It will return the number or rows affected

export(query, fname, sample=-1, filespath='./')  
fetchall()  
Fetch all (remaining) rows of a query result, returning them as a sequence of sequences (e.g. a list of tuples). Note that the cursor’s arraysize attribute can affect the performance of this operation.

An Error (or subclass) exception is raised if the previous call to .execute*() did not produce any result set or no call was issued yet.

fetchmany(size=None)  
Fetch the next set of rows of a query result, returning a sequence of sequences (e.g. a list of tuples). An empty sequence is returned when no more rows are available.

The number of rows to fetch per call is specified by the parameter. If it is not given, the cursor’s arraysize determines the number of rows to be fetched. The method should try to fetch as many rows as indicated
by the size parameter. If this is not possible due to the specified number of rows not being available, fewer rows may be returned.

An Error (or subclass) exception is raised if the previous call to .execute*() did not produce any result set or no call was issued yet.

Note there are performance considerations involved with the size parameter. For optimal performance, it is usually best to use the arraysize attribute. If the size parameter is used, then it is best for it to retain the same value from one .fetchmany() call to the next.

**fetchone()**

Fetch the next row of a query result set, returning a single sequence, or None when no more data is available.

**next()**

This method will make the cursor skip to the next available set, discarding any remaining rows from the current set.

If there are no more sets, the method returns None. Otherwise, it returns a true value and subsequent calls to the fetch methods will return rows from the next result set.

An Error (or subclass) exception is raised if the previous call to .execute*() did not produce any result set or no call was issued yet.

**scroll(value, mode='relative')**

Scroll the cursor in the result set to a new position according to mode.

If mode is 'relative' (default), value is taken as offset to the current position in the result set, if set to 'absolute', value states an absolute target position.

An IndexError is raised in case a scroll operation would leave the result set.

**setinputsizes(sizes)**

This method would be used before the .execute*() method is invoked to reserve memory. This implementation doesn’t use this.

**setoutputsize(size, column=None)**

Set a column buffer size for fetches of large columns This implementation doesn’t use this

### 3.2 Type conversion

functions for converting python objects to monetdb SQL format. If you want to add support for a specific type you should add a function as a value to the mapping dict and the datatype as key.

- **pymonetdb.sql.monetize.convert(data)**
  
  Return the appropriate conversion function based upon the python type.

- **pymonetdb.sql.monetize.monet_bool(data)**
  
  returns “true” or “false”

- **pymonetdb.sql.monetize.monet_bytes(data)**
  
  converts bytes to string

- **pymonetdb.sql.monetize.monet_date(data)**
  
  returns a casted date

- **pymonetdb.sql.monetize.monet_datetime(data)**
  
  returns a casted timestamp
pymonetdb.sql.monetize.monet_escape\(\text{data}\)
returns an escaped string

pymonetdb.sql.monetize.monet_none\(\_\)
returns a NULL string

pymonetdb.sql.monetize.monet_time\(\text{data}\)
returns a casted time

pymonetdb.sql.monetize.monet_timedelta\(\text{data}\)
returns timedelta casted to interval seconds

pymonetdb.sql.monetize.monet_unicode\(\text{data}\)
functions for converting monetdb SQL fields to Python objects

pymonetdb.sql.pythonize.Binary\(\text{data}\)
returns binary encoding of data

pymonetdb.sql.pythonize.DateFromTicks\(\text{ticks}\)
Convert ticks to python Date

pymonetdb.sql.pythonize.TimeFromTicks\(\text{ticks}\)
Convert ticks to python Time

pymonetdb.sql.pythonize.TimeTzFromTicks\(\text{ticks}\)
Convert ticks to python Time

pymonetdb.sql.pythonize.TimestampFromTicks\(\text{ticks}\)
Convert ticks to python Timestamp

pymonetdb.sql.pythonize.TimestampTzFromTicks\(\text{ticks}\)
Convert ticks to python Timestamp

pymonetdb.sql.pythonize.convert\(\text{data, type_code}\)
Calls the appropriate conversion function based upon the python type

pymonetdb.sql.pythonize.oid\(\text{data}\)
represents an object identifier

For now we will just return the string representation just like mclient does.

pymonetdb.sql.pythonize.py_bool\(\text{data}\)
return python boolean

pymonetdb.sql.pythonize.py_bytes\(\text{data}\)
Returns a bytes (py3) or string (py2) object representing the input blob.

pymonetdb.sql.pythonize.py_date\(\text{data}\)
Returns a python Date

pymonetdb.sql.pythonize.py_day_interval\(\text{data: str}\) → int
Returns a python number of days where data represents a value of MonetDB’s INTERVAL DAY type which resembles a stringified decimal.

pymonetdb.sql.pythonize.py_sec_interval\(\text{data: str}\) → datetime.timedelta
Returns a python TimeDelta where data represents a value of MonetDB’s INTERVAL SECOND type which resembles a stringified decimal.

pymonetdb.sql.pythonize.py_time\(\text{data}\)
returns a python Time

pymonetdb.sql.pythonize.py_timestamp\(\text{data}\)
Returns a python Timestamp
pymonetdb Documentation, Release 1.4.1

pymonetdb.sql.pythonize.py_timestamptz(data)
   Returns a python Timestamp where data contains a tz code

pymonetdb.sql.pythonize.py_timetz(data)
   returns a python Time where data contains a tz code

pymonetdb.sql.pythonize.strip(data)
   returns a python string, with chopped off quotes, and replaced escape characters

3.3 MAPI

This is the python implementation of the mapi protocol.

class pymonetdb.mapi.Connection
    Bases: object
        MAPI (low level MonetDB API) connection
            cmd(operation)
                put a mapi command on the line
            connect(database, username, password, language, hostname=None, port=None, unix_socket=None, connect_timeout=-1, handshake_options=None)
                setup connection to MAPI server
                    unix_socket is used if hostname is not defined.
            disconnect()
                disconnect from the monetdb server
            set_reply_size(size)
                Set the amount of rows returned by the server.
                    args: size: The number of rows

class pymonetdb.mapi.HandshakeOption(level, name, fallback, value)
    Bases: object
        Option that can be set during the MAPI handshake
            Should be sent as <name>=<val>, where <val> is value converted to int. The level is used to determine if the server supports this option. The fallback is a function-like object that can be called with the value (not converted to an integer) as a parameter. Field sent can be used to keep track of whether the option has been sent.

pymonetdb.mapi.handle_error(error)
    Return exception matching error code.
        args: error (str): error string, potentially containing mapi error code
        returns:
            tuple (Exception, formatted error): returns OperationalError if unknown error or no error code in string

3.4 MonetDB remote control

class pymonetdb.control.Control(hostname=None, port=50000, passphrase=None, unix_socket=None, connect_timeout=-1)
    Bases: object
Use this module to manage your MonetDB databases. You can create, start, stop, lock, unlock, destroy your databases and request status information.

**create** *(database_name)*
Initialises a new database or multiplexfunnel in the MonetDB Server. A database created with this command makes it available for use, however in maintenance mode (see pymonetdb lock).

**defaults** *

**destroy** *(database_name)*
Removes the given database, including all its data and logfiles. Once destroy has completed, all data is lost. Be careful when using this command.

**get** *(database_name)*
gets value for property for the given database, or retrieves all properties for the given database

**inherit** *(database_name, property_)*
unsets property, reverting to its inherited value from the default configuration for the given database

**kill** *(database_name)*
Kills the given database, if the MonetDB Database Server is running. Note: killing a database should only be done as last resort to stop a database. A database being killed may end up with data loss.

**lock** *(database_name)*
Puts the given database in maintenance mode. A database under maintenance can only be connected to by the DBA. A database which is under maintenance is not started automatically. Use the “release” command to bring the database back for normal usage.

**neighbours** *

**release** *(database_name)*
Brings back a database from maintenance mode. A released database is available again for normal use. Use the “lock” command to take a database under maintenance.

**rename** *(old, new)*

**set** *(database_name, property_, value)*
sets property to value for the given database for a list of properties, use `pymonetdb get all`

**start** *(database_name)*
Starts the given database, if the MonetDB Database Server is running.

**status** *(database_name=Boolean)*
Shows the state of a given glob-style database match, or all known if none given. Instead of the normal mode, a long and crash mode control what information is displayed.

**stop** *(database_name)*
 Stops the given database, if the MonetDB Database Server is running.

**pymonetdb.control isempty** *(result)*
raises an exception if the result is not empty

**pymonetdb.control.parse_statusline** *(line)*
parses a sabdb format status line. Support v1 and v2.

### 3.5 pymonetdb Exceptions

MonetDB Python API specific exceptions
exception pymonetdb.exceptions.DataError
    Bases: pymonetdb.exceptions.DatabaseError

    Exception raised for errors that are due to problems with the processed data like division by zero, numeric value outside of range, etc. It must be a subclass of DatabaseError.

exception pymonetdb.exceptions.DatabaseError
    Bases: pymonetdb.exceptions.Error

    Exception raised for errors that are related to the database. It must be a subclass of Error.

exception pymonetdb.exceptions.Error
    Bases: Exception

    Exception that is the base class of all other error exceptions. You can use this to catch all errors with one single `except` statement. Warnings are not considered errors and thus should not use this class as base. It must be a subclass of the Python StandardError (defined in the module exceptions).

exception pymonetdb.exceptions.IntegrityError
    Bases: pymonetdb.exceptions.DatabaseError

    Exception raised when the relational integrity of the database is affected, e.g. a foreign key check fails. It must be a subclass of DatabaseError.

exception pymonetdb.exceptions.InterfaceError
    Bases: pymonetdb.exceptions.Error

    Exception raised for errors that are related to the database interface rather than the database itself. It must be a subclass of Error.

exception pymonetdb.exceptions.InternalError
    Bases: pymonetdb.exceptions.DatabaseError

    Exception raised when the database encounters an internal error, e.g. the cursor is not valid anymore, the transaction is out of sync, etc. It must be a subclass of DatabaseError.

exception pymonetdb.exceptions.NotSupportedError
    Bases: pymonetdb.exceptions.DatabaseError

    Exception raised in case a method or database API was used which is not supported by the database, e.g. requesting a .rollback() on a connection that does not support transaction or has transactions turned off. It must be a subclass of DatabaseError.

exception pymonetdb.exceptions.OperationalError
    Bases: pymonetdb.exceptions.DatabaseError

    Exception raised for errors that are related to the database’s operation and not necessarily under the control of the programmer, e.g. an unexpected disconnect occurs, the data source name is not found, a transaction could not be processed, a memory allocation error occurred during processing, etc. It must be a subclass of DatabaseError.

exception pymonetdb.exceptions.ProgrammingError
    Bases: pymonetdb.exceptions.DatabaseError

    Exception raised for programming errors, e.g. table not found or already exists, syntax error in the SQL statement, wrong number of parameters specified, etc. It must be a subclass of DatabaseError.

exception pymonetdb.exceptions.Warning
    Bases: Exception

    Exception raised for important warnings like data truncations while inserting, etc. It must be a subclass of the Python StandardError (defined in the module exceptions).
4.1 Github

We maintain pymonetdb on github. If you have any problems with pymonetdb please raise an issue in the issue tracker. Even better is if you have a solution to problem! In that case you can make our live easier by following these steps:

- fork our repository on github
- Add a tests that will fail because of the problem
- Fix the problem
- Run the test suite again
- Commit to your repository
- Issue a github pull request.

Also we try to be as much pep8 compatible as possible, where possible and reasonable.

4.2 Test suite

pymonetdb comes with a test suite. This test suite verifies that the code actually works and makes development much easier. To run all tests please run from the source:

```bash
$ pip install tox
$ tox

* MAPIPORT - what port is MonetDB running? _50000_ by default
* TSTHOSTNAME - where is MonetDB running? _localhost_ by default
* TSTPASSPHRASE - what passphrase to test control command? _testdb_ by default
* TSTDB - what database to use for testing? _demo_ by default
* TSTUSERNAME - username, _monetdb_ by default
* TSTPASSWORD - password, _monetdb_ by default
```
Note that you first need to create and start a monetdb database. If you want to run the control tests you need to set a passphrase and enable remote control:

```
$ monetdb create demo
$ monetdb release demo
$ monetdbd set control=yes <path to dbfarm>
$ monetdbd set passphrase=testdb <path to dbfarm>
```
CHAPTER 5

Indices and tables

- genindex
- modindex
- search
Python Module Index

p
- pymonetdb.control, 12
- pymonetdb.exceptions, 13
- pymonetdb.mapi, 12
- pymonetdb.sql.monetize, 10
- pymonetdb.sql.pythonize, 11
Index

B
Binary() (in module pymonetdb.sql.pythonize), 11

C
close() (pymonetdb.sql.connections.Connection method), 8
close() (pymonetdb.sql.cursors.Cursor method), 9
cmd() (pymonetdb.mapi.Connection method), 12
command() (pymonetdb.sql.connections.Connection method), 8
commit() (pymonetdb.sql.connections.Connection method), 8
connect() (pymonetdb.mapi.Connection method), 12
Connection (class in pymonetdb.mapi), 12
Connection (class in pymonetdb.sql.connections), 7
Connection.DatabaseError, 7
Connection.DataError, 7
Connection.Error, 7
Connection.IntegrityError, 7
Connection.InterfaceError, 7
Connection.InternalError, 8
Connection.NotSupportedError, 8
Connection.OperationalError, 8
Connection.ProgrammingError, 8
Connection.Warning, 8
Control (class in pymonetdb.control), 12
convert() (in module pymonetdb.sql.monetize), 10
convert() (in module pymonetdb.sql.pythonize), 11
create() (pymonetdb.control.Control method), 13
Cursor (class in pymonetdb.sql.cursors), 9
cursor() (pymonetdb.sql.connections.Connection method), 9
default_cursor (pymonetdb.sql.connections.Connection attribute), 8
defaults() (pymonetdb.control.Control method), 13
destroy() (pymonetdb.control.Control method), 13
disconnect() (pymonetdb.mapi.Connection method), 12

E
Error, 14
execute() (pymonetdb.sql.connections.Connection method), 8
execute() (pymonetdb.sql.cursors.Cursor method), 9
execute() (pymonetdb.sql.cursors.Cursor method), 9
export() (pymonetdb.sql.cursors.Cursor method), 9

F
fetchall() (pymonetdb.sql.cursors.Cursor method), 9
fetchmany() (pymonetdb.sql.cursors.Cursor method), 9
fetchone() (pymonetdb.sql.cursors.Cursor method), 10

G
get() (pymonetdb.control.Control method), 13
gettimeout() (pymonetdb.sql.connections.Connection method), 8

H
handle_error() (in module pymonetdb.mapi), 12
HandshakeOption (class in pymonetdb.mapi), 12

I
inherit() (pymonetdb.control.Control method), 13
IntegrityError, 14
InterfaceError, 14
InternalError, 14
isempty() (in module pymonetdb.control), 13

K
kill() (pymonetdb.control.Control method), 13

L
lock() (pymonetdb.control.Control method), 13

M
monet_bool() (in module pymonetdb.sql.monetize), 10
monet_bytes() (in module pymonetdb.sql.monetize), 10
monet_date() (in module pymonetdb.sql.monetize), 10
monet_datetime() (in module pymonetdb.sql.monetize), 10
monet_escape() (in module pymonetdb.sql.monetize), 10
monet_none() (in module pymonetdb.sql.monetize), 11
monet_time() (in module pymonetdb.sql.monetize), 11
monet_timedelta() (in module pymonetdb.sql.monetize), 11
monet_unicode() (in module pymonetdb.sql.monetize), 11

N
neighbours() (pymonetdb.control.Control method), 13
next() (pymonetdb.sql.cursors.Cursor method), 10
nextset() (pymonetdb.sql.cursors.Cursor method), 10
NotSupportedError, 14

O
oid() (in module pymonetdb.sql.pythonize), 11
OperationalError, 14

P
parse_statusline() (in module pymonetdb.control), 13
ProgrammingError, 14
py_bool() (in module pymonetdb.sql.pythonize), 11
py_bytes() (in module pymonetdb.sql.pythonize), 11
py_date() (in module pymonetdb.sql.pythonize), 11
py_day_interval() (in module pymonetdb.sql.pythonize), 11
py_sec_interval() (in module pymonetdb.sql.pythonize), 11
py_time() (in module pymonetdb.sql.pythonize), 11
py_timestamp() (in module pymonetdb.sql.pythonize), 11
py_timestamptz() (in module pymonetdb.sql.pythonize), 11
py_timez() (in module pymonetdb.sql.pythonize), 12
pymonetdb.control (module), 12
pymonetdb.exceptions (module), 12
pymonetdb.mapi (module), 12
pymonetdb.sql.monetize (module), 10
pymonetdb.sql.pythonize (module), 11

R
release() (pymonetdb.control.Control method), 13
rename() (pymonetdb.control.Control method), 13
rollback() (pymonetdb.sql.connections.Connection method), 9

S
scroll() (pymonetdb.sql.cursors.Cursor method), 10
set() (pymonetdb.control.Control method), 13
set_autocommit() (pymonetdb.sql.connections.Connection method), 9
set_reply_size() (pymonetdb.mapi.Connection method), 12
set_replay_size() (pymonetdb.mapi.Connection method), 9
set_replysize() (pymonetdb.sql.connections.Connection method), 9
set_sizeheader() (pymonetdb.sql.connections.Connection method), 9
set_timezone() (pymonetdb.sql.connections.Connection method), 9
setinputsizes() (pymonetdb.sql.cursors.Cursor method), 10
setoutputsize() (pymonetdb.sql.cursors.Cursor method), 10
settimeout() (pymonetdb.sql.connections.Connection method), 9
start() (pymonetdb.control.Control method), 13
status() (pymonetdb.control.Control method), 13
stop() (pymonetdb.control.Control method), 13
strip() (in module pymonetdb.sql.pythonize), 12

T
TimeFromTicks() (in module pymonetdb.sql.pythonize), 11
TimestampFromTicks() (in module pymonetdb.sql.pythonize), 11
TimestampTzFromTicks() (in module pymonetdb.sql.pythonize), 11
TimeTzFromTicks() (in module pymonetdb.sql.pythonize), 11
W
Warning, 14