rexpro Documentation

Release 0.4.5

Cody Lee

November 05, 2015

Contents

rexpro is an experimental RexPro socket interface for python

CHAPTER 1

Features

- Straightforward syntax
- Connection Pooling Support
- Gevent Compatible - see `rexpro.connectors.rgevent`
- Eventlet Compatible - see `rexpro.connectors.reventlet`
- **Tested against Celery**
  - Only Gevent has been tested (must use `-P gevent` argument)
  - Testing needed against default prefork method, and eventlet

CHAPTER 2

Links

- Documentation: http://rexpro.readthedocs.org/
- Official Repository: https://github.com/platinummonkey/rexpro.git
- Package: https://pypi.python.org/pypi/rexpro/

rexpro is known to support Python 2.7. Python 3.x is not compatible due to the use of gevent.

CHAPTER 3

Download

PyPI: https://pypi.python.org/pypi/rexpro/

```
$ pip install rexpro
```

Source: https://github.com/platinummonkey/rexpro-python.git

```
$ git clone git://github.com/platinummonkey/rexpro-python.git
$ python setup.py install
```

CHAPTER 4

Contents

4.1 Quick Start

4.1.1 Usage

Note: This section provides a quick summary of rexpro features. A more detailed listing is available in the full documentations.

4.1.2 Setup Connection

You'll need to setup the connection to the graph database.

```
from rexpro import RexProConnection

# Without Authentication
conn = RexProConnection('localhost', 8184, 'graph')
# With authentication
#conn = RexProConnection('localhost', 8184, 'graph', username='rexster', password='rexster')
```

4.1.3 Basic Query

We will run a simple query using a groovy script with the supplied params

```
#create and return some elements
#execute takes a script, and optionally, parameter bindings
#for parameterized queries
elements = conn.execute(
    """
    def v1 = g.addVertex([prop:bound_param1])
    def v2 = g.addVertex([prop:bound_param2])
    def e = g.addEdge(v1, v2, 'connects', [prop:bound_param3])
    return [v1, v2, e]
    """,
    {'bound_param1':'b1', 'bound_param2':'b2', 'bound_param3':'b3'}
)

#the contents of elements will be:
({'_id': '0', '_properties': {'prop': 'b1'}, '_type': 'vertex'},
```

```
 {'_id': '1', '_properties': {'prop': 'b2'}, '_type': 'vertex'},
 {'_id': '2',
  '_inV': '1',
  '_label': 'connects',
  '_outV': '0',
  '_properties': {'prop': 'b3'},
  '_type': 'edge'})
```

4.1.4 Transactional Query

If you're using this with a transactional graph you can do requests in the context of the transaction one of two ways

Explicit

```
conn.test_connection()  # We test the connection here and attempt to re-connect if necessary
conn.open_transaction()
try:
   conn.execute("// do something")
   conn.execute("// do another thing")
except:
   conn.close_transaction(False)  # rollback
   # Handle the exception here
else:
   conn.close_transaction(True)   # commit
```

If something fails mid-transaction, you're responsible yourself for handling the failure and properly rolling back the transaction on the server end (i.e. call *conn.close_transaction(False)*).

Context Manager

```
with conn.transaction():
    conn.execute("// do something")
    conn.execute("// do another thing")
```

If an exception occurs inside the transaction context (i.e. when one of the scripts fail to execute), the entire transaction will be rolled back on the server side and it will be as if none of the scripts ever took place. The exception will be reraised, so you can catch it outside of the with-block and handle it properly.

4.1.5 Query Scoping & Global Variables

A RexPro connection is basically a connection to a gremlin REPL. Queries executed with the RexProConnection's `execute` method are automatically wrapped in a closure before being executed to avoid cluttering the global namespace with variables defined in previous queries. A globally available `g` graph object is is automatically defined at the beginning of a RexPro session.

If you would like to define additional global variables, don't define variables with a `def` statement. For example:

```
#number will become a global variable for this session
conn.execute("number = 5")

#another_number is only available for this query
conn.execute("def another_number = 6")
```

4.2 Reference

4.2.1 Connection

This is left purely for backwards compatibility

The defaults are now:

Class Name	Alias to
`RexProSocket`	`rexpro.connectors.sync.RexProSyncSocket`
`RexProConnection`	`rexpro.connectors.sync.RexProSyncConnection`
`RexProConnectionPool`	`rexpro.connectors.sync.RexProSyncConnectionPool`

See *Synchronous Connectors*.

4.2.2 Connectors

Synchronous

class `rexpro.connectors.sync.connection.`**`RexProSyncConnection`**(*host*, *port*, *graph_name*, *graph_obj_name='g'*, *username=''*, *password=''*, *timeout=None*, *session_key=None*, *pool_session=None*)

Synchronous RexProConnection

`SOCKET_CLASS`
alias of `RexProSyncSocket`

`close`(*soft=False*)
Close a connection

Parameters **soft** (*bool*) – Softly close the connection - do not actually close the socket (default: False)

`close_transaction`(*success=True*)
closes an open transaction

Parameters **success** (*bool*) – indicates which status to close the transaction with, True will commit the changes, False will roll them back

`execute`(*script*, *params=None*, *isolate=True*, *transaction=True*, *language='groovy'*)
executes the given gremlin script with the provided parameters

Parameters

- **script** (*str*) – the gremlin script to isolate
- **params** (*dictionary*) – the parameters to execute the script with
- **isolate** (*bool*) – wraps the script in a closure so any variables set aren't persisted for the next execute call
- **transaction** (*bool*) – query will be wrapped in a transaction if set to True (default)
- **language** (*str*) – the script language that should be used (defaults to groovy)

Return type list

open(*soft=False*)
open the connection to the database

Parameters **soft** (*bool*) – Attempt to re-use the connection, if False (default), create a new socket

open_transaction()
opens a transaction

test_connection()
Test the socket, if it's errored or closed out, try to reconnect. Otherwise raise and Exception

transaction(**args*, ***kwds*)
Context manager that opens a transaction and closes it at the end of it's code block, use with the 'with' statement

Example:

```
conn = RexproSyncConnection(host, port, graph_name)
with conn.transaction():
    results = conn.execute(script, params)
```

class rexpro.connectors.sync.connection.**RexProSyncConnectionPool**(*host*, *port*, *graph_name*, *graph_obj_name='g'*, *username=''*, *password=''*, *timeout=None*, *pool_size=10*, *with_session=False*)

Synchronous RexProConnectionPool

CONN_CLASS
alias of RexProSyncConnection

QUEUE_CLASS
alias of Queue

close_all(*force_commit=False*)
Close all pool connections for a clean shutdown

close_connection(*conn*, *soft=False*)
Close a connection and restore it to the pool

Parameters

- **conn** (*RexProConnection*) – a rexpro connection that was pull from the Pool
- **soft** (*bool*) – define whether to soft-close the connection or hard-close the socket

connection(**args*, ***kwds*)
Context manager that conveniently grabs a connection from the pool and provides it with the context cleanly closes up the connection and restores it to the pool afterwards

Parameters

- **host** (*str (ip address)*) – the rexpro server to connect to
- **port** (*int*) – the rexpro server port to connect to
- **graph_name** (*str*) – the graph to connect to
- **graph_obj_name** (*str*) – The graph object to use

- **username** (*str*) – the username to use for authentication (optional)
- **password** (*str*) – the password to use for authentication (optional)

create_connection(**args*, ***kwargs*)
Get a connection from the pool if available, otherwise return a new connection if the pool isn't full

Parameters

- **host** (*str (ip address)*) – the rexpro server to connect to
- **port** (*int*) – the rexpro server port to connect to
- **graph_name** (*str*) – the graph to connect to
- **graph_obj_name** (*str*) – The graph object to use
- **username** (*str*) – the username to use for authentication (optional)
- **password** (*str*) – the password to use for authentication (optional)

Return type RexProConnection

get(**args*, ***kwargs*)
Retrieve a rexpro connection from the pool

Parameters

- **host** (*str (ip address)*) – the rexpro server to connect to
- **port** (*int*) – the rexpro server port to connect to
- **graph_name** (*str*) – the graph to connect to
- **graph_obj_name** (*str*) – The graph object to use
- **username** (*str*) – the username to use for authentication (optional)
- **password** (*str*) – the password to use for authentication (optional)

Return type RexProConnection

put(*conn*)
Restore a connection to the pool

Parameters **conn** (*RexProSyncConnection | RexProGeventConnection | RexProEventletConnection | RexProConnection*) – A rexpro connection to restore to the pool

class rexpro.connectors.sync.connection.**RexProSyncSocket**(*family=2*, *type=1*, *proto=0*, *_sock=None*)
Subclass of python's socket that sends and received rexpro messages

inherits from socket.socket

accept() -> (*socket object*, *address info*)
Wait for an incoming connection. Return a new socket representing the connection, and the address of the client. For IP sockets, the address info is a pair (hostaddr, port).

bind(*address*)
Bind the socket to a local address. For IP sockets, the address is a pair (host, port); the host must refer to the local host. For raw packet sockets the address is a tuple (ifname, proto [,pkttype [,hatype]])

close()
Close the socket. It cannot be used after this call.

connect(*address*)
Connect the socket to a remote address. For IP sockets, the address is a pair (host, port).

connect_ex(*address*) → errno
This is like connect(address), but returns an error code (the errno value) instead of raising an exception when an error occurs.

dup() → socket object
Return a new socket object connected to the same system resource.

family
the socket family

fileno() → integer
Return the integer file descriptor of the socket.

get_response()
gets the message type and message from rexster

Basic Message Structure: reference: https://github.com/tinkerpop/rexster/wiki/RexPro-Messages

segment	type (bytes)	description
protocol version	byte (1)	Version of RexPro, should be 1
serializer type	byte (1)	Type of Serializer: msgpack==0, json==1
reserved for future	byte (4)	Reserved for future use.
message type	byte (1)	Tye type of message as described in the value columns.
message size	int (4)	The length of the message body
message body	byte (n)	The body of the message itself. The Good, Bad and Ugly.

Message Types:

message type	type	value	description
session	request	1	A request to open or close the session with the RexPro Server
session	response	2	RexPro server response to session request
script	request	3	A request to process a gremlin script
script	response	5	A response to a script request
error	response	0	A RexPro server error response

Returns RexProMessage

getpeername() → address info
Return the address of the remote endpoint. For IP sockets, the address info is a pair (hostaddr, port).

getsockname() → address info
Return the address of the local endpoint. For IP sockets, the address info is a pair (hostaddr, port).

getsockopt(*level*, *option*[, *buffersize*]) → value
Get a socket option. See the Unix manual for level and option. If a nonzero buffersize argument is given, the return value is a string of that length; otherwise it is an integer.

gettimeout() → timeout
Returns the timeout in seconds (float) associated with socket operations. A timeout of None indicates that timeouts on socket operations are disabled.

listen(*backlog*)
Enable a server to accept connections. The backlog argument must be at least 0 (if it is lower, it is set to 0); it specifies the number of unaccepted connections that the system will allow before refusing new connections.

makefile([*mode*[, *bufsize*]]) → file object
Return a regular file object corresponding to the socket. The mode and bufsize arguments are as for the built-in open() function.

proto
the socket protocol

recv

recv_into

recvfrom

recvfrom_into

send

send_message(*msg*)
Serializes the given message and sends it to rexster

Parameters **msg** (*RexProMessage*) – the message instance to send to rexster

sendall(*data*[, *flags*])
Send a data string to the socket. For the optional flags argument, see the Unix manual. This calls send() repeatedly until all data is sent. If an error occurs, it's impossible to tell how much data has been sent.

sendto

setblocking(*flag*)
Set the socket to blocking (flag is true) or non-blocking (false). setblocking(True) is equivalent to settimeout(None); setblocking(False) is equivalent to settimeout(0.0).

setsockopt(*level*, *option*, *value*)
Set a socket option. See the Unix manual for level and option. The value argument can either be an integer or a string.

settimeout(*timeout*)
Set a timeout on socket operations. 'timeout' can be a float, giving in seconds, or None. Setting a timeout of None disables the timeout feature and is equivalent to setblocking(1). Setting a timeout of zero is the same as setblocking(0).

shutdown(*flag*)
Shut down the reading side of the socket (flag == SHUT_RD), the writing side of the socket (flag == SHUT_WR), or both ends (flag == SHUT_RDWR).

type
the socket type

Gevent

class rexpro.connectors.rgevent.connection.**RexProGeventConnection**(*host*, *port*, *graph_name*, *graph_obj_name='g'*, *username=''*, *password=''*, *timeout=None*, *session_key=None*, *pool_session=None*)
Gevent-based RexProConnection

SOCKET_CLASS
alias of RexProGeventSocket

close(*soft=False*)
Close a connection

Parameters **soft** (*bool*) – Softly close the connection - do not actually close the socket (default: False)

close_transaction(*success=True*)
closes an open transaction

Parameters **success** (*bool*) – indicates which status to close the transaction with, True will commit the changes, False will roll them back

execute(*script*, *params=None*, *isolate=True*, *transaction=True*, *language='groovy'*)
executes the given gremlin script with the provided parameters

Parameters

- **script** (*str*) – the gremlin script to isolate
- **params** (*dictionary*) – the parameters to execute the script with
- **isolate** (*bool*) – wraps the script in a closure so any variables set aren't persisted for the next execute call
- **transaction** (*bool*) – query will be wrapped in a transaction if set to True (default)
- **language** (*str*) – the script language that should be used (defaults to groovy)

Return type list

open(*soft=False*)
open the connection to the database

Parameters **soft** (*bool*) – Attempt to re-use the connection, if False (default), create a new socket

open_transaction()
opens a transaction

test_connection()
Test the socket, if it's errored or closed out, try to reconnect. Otherwise raise and Exception

transaction(**args*, ***kwds*)
Context manager that opens a transaction and closes it at the end of it's code block, use with the 'with' statement

Example:

```
conn = RexproSyncConnection(host, port, graph_name)
with conn.transaction():
    results = conn.execute(script, params)
```

class rexpro.connectors.rgevent.connection.**RexProGeventConnectionPool**(*host*, *port*, *graph_name*, *graph_obj_name='g'*, *username=''*, *password=''*, *timeout=None*, *pool_size=10*, *with_session=False*)

Gevent-based RexProConnectionPool

CONN_CLASS
alias of RexProGeventConnection

QUEUE_CLASS
alias of Queue

close_all(*force_commit=False*)
Close all pool connections for a clean shutdown

close_connection(*conn*, *soft=False*)
Close a connection and restore it to the pool

Parameters

- **conn** (*RexProConnection*) – a rexpro connection that was pull from the Pool
- **soft** (*bool*) – define whether to soft-close the connection or hard-close the socket

connection(**args*, ***kwds*)
Context manager that conveniently grabs a connection from the pool and provides it with the context cleanly closes up the connection and restores it to the pool afterwards

Parameters

- **host** (*str (ip address)*) – the rexpro server to connect to
- **port** (*int*) – the rexpro server port to connect to
- **graph_name** (*str*) – the graph to connect to
- **graph_obj_name** (*str*) – The graph object to use
- **username** (*str*) – the username to use for authentication (optional)
- **password** (*str*) – the password to use for authentication (optional)

create_connection(**args*, ***kwargs*)
Get a connection from the pool if available, otherwise return a new connection if the pool isn't full

Parameters

- **host** (*str (ip address)*) – the rexpro server to connect to
- **port** (*int*) – the rexpro server port to connect to
- **graph_name** (*str*) – the graph to connect to
- **graph_obj_name** (*str*) – The graph object to use
- **username** (*str*) – the username to use for authentication (optional)
- **password** (*str*) – the password to use for authentication (optional)

Return type RexProConnection

get(**args*, ***kwargs*)
Retrieve a rexpro connection from the pool

Parameters

- **host** (*str (ip address)*) – the rexpro server to connect to
- **port** (*int*) – the rexpro server port to connect to
- **graph_name** (*str*) – the graph to connect to
- **graph_obj_name** (*str*) – The graph object to use
- **username** (*str*) – the username to use for authentication (optional)
- **password** (*str*) – the password to use for authentication (optional)

Return type RexProConnection

put(*conn*)
Restore a connection to the pool

Parameters **conn** (*RexProSyncConnection | RexProGeventConnection | RexProEventletConnection | RexProConnection*) – A rexpro connection to restore to the pool

class rexpro.connectors.rgevent.connection.**RexProGeventSocket**(*family=2*, *type=1*, *proto=0*, *_sock=None*)
Subclass of gevent's socket that sends and received rexpro messages

inherits from gevent.socket.socket

accept()

bind(*address*)
Bind the socket to a local address. For IP sockets, the address is a pair (host, port); the host must refer to the local host. For raw packet sockets the address is a tuple (ifname, proto [,pkttype [,hatype]])

close(*_closedsocket=<class 'gevent.socket._closedsocket'>*, *cancel_wait_ex=error(9, 'File descriptor was closed in another greenlet')*)

closed

connect(*address*)

connect_ex(*address*)

dup() → socket object
Return a new socket object connected to the same system resource. Note, that the new socket does not inherit the timeout.

family
the socket family

fileno() → integer
Return the integer file descriptor of the socket.

get_response()
gets the message type and message from rexster

Basic Message Structure: reference: https://github.com/tinkerpop/rexster/wiki/RexPro-Messages

segment	type (bytes)	description
protocol version	byte (1)	Version of RexPro, should be 1
serializer type	byte (1)	Type of Serializer: msgpack==0, json==1
reserved for future	byte (4)	Reserved for future use.
message type	byte (1)	Tye type of message as described in the value columns.
message size	int (4)	The length of the message body
message body	byte (n)	The body of the message itself. The Good, Bad and Ugly.

Message Types:

message type	type	value	description
session	request	1	A request to open or close the session with the RexPro Server
session	response	2	RexPro server response to session request
script	request	3	A request to process a gremlin script
script	response	5	A response to a script request
error	response	0	A RexPro server error response

Returns RexProMessage

getpeername() → address info
Return the address of the remote endpoint. For IP sockets, the address info is a pair (hostaddr, port).

getsockname() → address info
Return the address of the local endpoint. For IP sockets, the address info is a pair (hostaddr, port).

getsockopt(*level*, *option*[, *buffersize*]) → value
Get a socket option. See the Unix manual for level and option. If a nonzero buffersize argument is given, the return value is a string of that length; otherwise it is an integer.

gettimeout()

listen(*backlog*)
Enable a server to accept connections. The backlog argument must be at least 0 (if it is lower, it is set to 0); it specifies the number of unaccepted connections that the system will allow before refusing new connections.

makefile(*mode='r'*, *bufsize=-1*)

proto
the socket protocol

recv(**args*)

recv_into(**args*)

recvfrom(**args*)

recvfrom_into(**args*)

ref

send(*data*, *flags=0*, *timeout=<object object at 0x7f85a016fd00>*)

send_message(*msg*)
Serializes the given message and sends it to rexster

Parameters **msg** (*RexProMessage*) – the message instance to send to rexster

sendall(*data*, *flags=0*)

sendto(**args*)

setblocking(*flag*)

setsockopt(*level*, *option*, *value*)
Set a socket option. See the Unix manual for level and option. The value argument can either be an integer or a string.

settimeout(*howlong*)

shutdown(*how*)

type
the socket type

Eventlet

class rexpro.connectors.reventlet.connection.**RexProEventletConnection**(*host*, *port*, *graph_name*, *graph_obj_name='g'*, *username=''*, *password=''*, *timeout=None*, *session_key=None*, *pool_session=None*)
Eventlet-based RexProConnection

SOCKET_CLASS
alias of RexProEventletSocket

close(*soft=False*)
Close a connection

Parameters **soft** (*bool*) – Softly close the connection - do not actually close the socket (default: False)

close_transaction(*success=True*)
closes an open transaction

Parameters **success** (*bool*) – indicates which status to close the transaction with, True will commit the changes, False will roll them back

execute(*script*, *params=None*, *isolate=True*, *transaction=True*, *language='groovy'*)
executes the given gremlin script with the provided parameters

Parameters

- **script** (*str*) – the gremlin script to isolate
- **params** (*dictionary*) – the parameters to execute the script with
- **isolate** (*bool*) – wraps the script in a closure so any variables set aren't persisted for the next execute call
- **transaction** (*bool*) – query will be wrapped in a transaction if set to True (default)
- **language** (*str*) – the script language that should be used (defaults to groovy)

Return type list

open(*soft=False*)
open the connection to the database

Parameters **soft** (*bool*) – Attempt to re-use the connection, if False (default), create a new socket

open_transaction()
opens a transaction

test_connection()
Test the socket, if it's errored or closed out, try to reconnect. Otherwise raise and Exception

transaction(**args*, ***kwds*)
Context manager that opens a transaction and closes it at the end of it's code block, use with the 'with' statement

Example:

```
conn = RexproSyncConnection(host, port, graph_name)
with conn.transaction():
    results = conn.execute(script, params)
```

class rexpro.connectors.reventlet.connection.**RexProEventletConnectionPool**(*host*, *port*, *graph_name*, *graph_obj_name='g'*, *username=''*, *password=''*, *timeout=None*, *pool_size=10*, *with_session=False*)

Eventlet-based RexProConnectionPool

CONN_CLASS
alias of RexProEventletConnection

QUEUE_CLASS
alias of Queue

close_all(*force_commit=False*)
Close all pool connections for a clean shutdown

close_connection(*conn*, *soft=False*)
Close a connection and restore it to the pool

Parameters

- **conn** (*RexProConnection*) – a rexpro connection that was pull from the Pool
- **soft** (*bool*) – define whether to soft-close the connection or hard-close the socket

connection(**args*, ***kwds*)
Context manager that conveniently grabs a connection from the pool and provides it with the context cleanly closes up the connection and restores it to the pool afterwards

Parameters

- **host** (*str (ip address)*) – the rexpro server to connect to

- **port** (*int*) – the rexpro server port to connect to
- **graph_name** (*str*) – the graph to connect to
- **graph_obj_name** (*str*) – The graph object to use
- **username** (*str*) – the username to use for authentication (optional)
- **password** (*str*) – the password to use for authentication (optional)

create_connection(**args*, ***kwargs*)

Get a connection from the pool if available, otherwise return a new connection if the pool isn't full

Parameters

- **host** (*str (ip address)*) – the rexpro server to connect to
- **port** (*int*) – the rexpro server port to connect to
- **graph_name** (*str*) – the graph to connect to
- **graph_obj_name** (*str*) – The graph object to use
- **username** (*str*) – the username to use for authentication (optional)
- **password** (*str*) – the password to use for authentication (optional)

Return type RexProConnection

get(**args*, ***kwargs*)

Retrieve a rexpro connection from the pool

Parameters

- **host** (*str (ip address)*) – the rexpro server to connect to
- **port** (*int*) – the rexpro server port to connect to
- **graph_name** (*str*) – the graph to connect to
- **graph_obj_name** (*str*) – The graph object to use
- **username** (*str*) – the username to use for authentication (optional)
- **password** (*str*) – the password to use for authentication (optional)

Return type RexProConnection

put(*conn*)

Restore a connection to the pool

Parameters **conn** (*RexProSyncConnection* | *RexProGeventConnection* | *RexProEventletConnection* | *RexProConnection*) – A rexpro connection to restore to the pool

class rexpro.connectors.reventlet.connection.**RexProEventletSocket**(*family_or_realsock=2*, **args*, ***kwargs*)

Subclass of eventlet's socket that sends and received rexpro messages

inherits from eventlet.green.socket.socket

accept()

connect(*address*)

connect_ex(*address*)

dup(**args*, ***kw*)

get_response()
gets the message type and message from rexster

Basic Message Structure: reference: https://github.com/tinkerpop/rexster/wiki/RexPro-Messages

segment	type (bytes)	description
protocol version	byte (1)	Version of RexPro, should be 1
serializer type	byte (1)	Type of Serializer: msgpack==0, json==1
reserved for future	byte (4)	Reserved for future use.
message type	byte (1)	Tye type of message as described in the value columns.
message size	int (4)	The length of the message body
message body	byte (n)	The body of the message itself. The Good, Bad and Ugly.

Message Types:

message type	type	value	description
session	request	1	A request to open or close the session with the RexPro Server
session	response	2	RexPro server response to session request
script	request	3	A request to process a gremlin script
script	response	5	A response to a script request
error	response	0	A RexPro server error response

Returns RexProMessage

gettimeout()

makeGreenFile(**args*, ***kw*)

makefile(**args*, ***kw*)

recv(*buflen*, *flags=0*)

recv_into(**args*)

recvfrom(**args*)

recvfrom_into(**args*)

send(*data*, *flags=0*)

send_message(*msg*)
Serializes the given message and sends it to rexster

Parameters **msg** (*RexProMessage*) – the message instance to send to rexster

sendall(*data*, *flags=0*)

sendto(**args*)

setblocking(*flag*)

settimeout(*howlong*)

4.2.3 Messages

class rexpro.messages.**ErrorResponse**(*meta*, *message*, *data=None*, ***kwargs*)

AUTH_FAILURE_ERROR = 3

CHANNEL_CONFIG_ERROR = 5

GRAPH_CONFIG_ERROR = 4

INVALID_MESSAGE_ERROR = 0

INVALID_SESSION_ERROR = 1

MESSAGE_TYPE = None

RESULT_SERIALIZATION_ERROR = 6

SCRIPT_FAILURE_ERROR = 2

classmethod deserialize(*data*)

get_message_list()
Creates and returns the list containing the data to be serialized into a message

get_meta()
Returns a dictionary of message meta data depending on other set values

static interpret_response(*response*)
interprets the response from rexster, returning the relevant response message object

raise_exception()

serialize()
Serializes this message to send to rexster

The format as far as I can tell is this:

type (bytes)	description
byte(1)	Message type
byte(4)	Message Length
byte(n)	msgpack serialized message

the actual message is just a list of values, all seem to start with version, session, and a unique request id the session and unique request id are uuid bytes, and the version and are each 1 byte unsigned integers

class rexpro.messages.**MessageTypes**
Enumeration of RexPro send message types

ERROR = 0

SCRIPT_REQUEST = 3

SCRIPT_RESPONSE = 5

SESSION_REQUEST = 1

SESSION_RESPONSE = 2

class rexpro.messages.**MsgPackScriptResponse**(*results*, *bindings*, ***kwargs*)

MESSAGE_TYPE = None

classmethod deserialize(*data*)

get_message_list()
Creates and returns the list containing the data to be serialized into a message

get_meta()
Returns a dictionary of message meta data depending on other set values

static interpret_response(*response*)
interprets the response from rexster, returning the relevant response message object

serialize()
Serializes this message to send to rexster

The format as far as I can tell is this:

type (bytes)	description
byte(1)	Message type
byte(4)	Message Length
byte(n)	msgpack serialized message

the actual message is just a list of values, all seem to start with version, session, and a unique request id the session and unique request id are uuid bytes, and the version and are each 1 byte unsigned integers

class rexpro.messages.**RexProMessage**
Base class for rexpro message types

MESSAGE_TYPE = None

classmethod deserialize(*data*)
Constructs a message instance from the given data

Parameters **data** (*str/bytearray*) – the raw data, minus the type and size info, from rexster

Return type RexProMessage

get_message_list()
Creates and returns the list containing the data to be serialized into a message

get_meta()
Returns a dictionary of message meta data depending on other set values

static interpret_response(*response*)
interprets the response from rexster, returning the relevant response message object

serialize()
Serializes this message to send to rexster

The format as far as I can tell is this:

type (bytes)	description
byte(1)	Message type
byte(4)	Message Length
byte(n)	msgpack serialized message

the actual message is just a list of values, all seem to start with version, session, and a unique request id the session and unique request id are uuid bytes, and the version and are each 1 byte unsigned integers

class rexpro.messages.**ScriptRequest**(*script*, *params=None*, *session_key=None*, *graph_name=None*, *graph_obj_name=None*, *in_session=True*, *isolate=True*, *in_transaction=True*, *language='groovy'*, ***kwargs*)
Message that executes a gremlin script and returns the response

class Language

GROOVY = 'groovy'

JAVA = 'java'

PYTHON = 'python'

SCALA = 'scala'

ScriptRequest.**MESSAGE_TYPE = 3**

classmethod ScriptRequest.**deserialize**(*data*)
Constructs a message instance from the given data

Parameters **data** (*str/bytearray*) – the raw data, minus the type and size info, from rexster

Return type RexProMessage

ScriptRequest.**get_message_list**()

ScriptRequest.**get_meta**()

static ScriptRequest.**interpret_response**(*response*)
interprets the response from rexster, returning the relevant response message object

ScriptRequest.**serialize**()
Serializes this message to send to rexster

The format as far as I can tell is this:

type (bytes)	description
byte(1)	Message type
byte(4)	Message Length
byte(n)	msgpack serialized message

the actual message is just a list of values, all seem to start with version, session, and a unique request id the session and unique request id are uuid bytes, and the version and are each 1 byte unsigned integers

ScriptRequest.**serialize_parameters**()
returns a serialization of the supplied parameters

class rexpro.messages.**SessionRequest**(*graph_name=None*, *graph_obj_name=None*, *username=''*, *password=''*, *session_key=None*, *kill_session=False*, **args*, ***kwargs*)
Message for creating a session with rexster

MESSAGE_TYPE = 1

classmethod **deserialize**(*data*)
Constructs a message instance from the given data

Parameters **data** (*str/bytearray*) – the raw data, minus the type and size info, from rexster

Return type RexProMessage

get_message_list()
Constructs a Session Request Message List

field	type	description
Session	byte (16)	The UUID for the Session. Set each byte to zero for an "empty" session.
Request	byte (16)	The UUID for the request. Uniquely identifies the request from the client. Get's passed back and forth so the response can be mapped to a request.
Meta	Map	Message specific properties described below
Username	String	The username to access the RexPro Server assuming auth is turned on. Ignored if no auth.
Password	String	The password to access the RexPro Server assuming auth is turned on. Ignored if no auth.

Meta Attributes:

field	type	description
graphName	String	The name of the graph to open a session on. Optional
graphObjName	String	The variable name of the Graph object, defaults to *g*. Optional
killSession	Bool	If true, the given session will be destroyed, else one will be created, default False

get_meta()

static interpret_response(*response*)
interprets the response from rexster, returning the relevant response message object

serialize()
Serializes this message to send to rexster

The format as far as I can tell is this:

type (bytes)	description
byte(1)	Message type
byte(4)	Message Length
byte(n)	msgpack serialized message

the actual message is just a list of values, all seem to start with version, session, and a unique request id the session and unique request id are uuid bytes, and the version and are each 1 byte unsigned integers

class rexpro.messages.**SessionResponse**(*session_key*, *meta*, *languages*, ***kwargs*)

MESSAGE_TYPE = None

classmethod deserialize(*data*)

get_message_list()
Creates and returns the list containing the data to be serialized into a message

get_meta()
Returns a dictionary of message meta data depending on other set values

static interpret_response(*response*)
interprets the response from rexster, returning the relevant response message object

serialize()
Serializes this message to send to rexster

The format as far as I can tell is this:

type (bytes)	description
byte(1)	Message type
byte(4)	Message Length
byte(n)	msgpack serialized message

the actual message is just a list of values, all seem to start with version, session, and a unique request id the session and unique request id are uuid bytes, and the version and are each 1 byte unsigned integers

rexpro.messages.**bytearray_to_text**(*data*)

rexpro.messages.**int_from_32bit_array**(*val*)
Converts an integer from a 32 bit bytearray

Parameters **val** (*int*) – the value to convert to an int

Return type int

rexpro.messages.**int_to_32bit_array**(*val*)
Converts an integer to 32 bit bytearray

Parameters **val** (*int*) – the value to convert to bytes

Return type bytearray

4.2.4 Exceptions

exception rexpro.exceptions.**RexProAuthenticationFailure**
Invalid authentication credentials provided

args

message

exception rexpro.exceptions.**RexProChannelConfigException**
Channel Configuration error

This is a serious problem.

args

message

exception rexpro.exceptions.**RexProConnectionException**
Raised when there are problems with the rexster connection

args

message

exception rexpro.exceptions.**RexProException**
Base RexProException

args

message

exception rexpro.exceptions.**RexProGraphConfigException**
Graph Configuration error

This is a serious problem.

args

message

exception rexpro.exceptions.**RexProInvalidConnectorTypeException**
Raised when the specified connector type isn't supported

args

message

exception rexpro.exceptions.**RexProInvalidMessageException**
Invalid message was provided, This may be an incompatible msgpack problem

args

message

exception rexpro.exceptions.**RexProInvalidSessionException**
An invalid or expired sessions was provided

args

message

exception rexpro.exceptions.**RexProResponseException**
Generic Exception Message Response

args

message

exception rexpro.exceptions.**RexProScriptException**
Raised when there's an error with a script request

args

message

exception rexpro.exceptions.**RexProSerializationException**
Serialization error, check your Titan Version compatibility

args

message

4.2.5 Utils

rexpro.utils.**get_rexpro**(*stype='sync'*)
Obtain the RexPro Socket, Connection and Connection Pool classes for the desired application

Options include:

- 'sync' - Default, Synchronous python sockets
- 'gevent' - with gevent concurrency
- 'eventlet' - with eventlet concurrency

Example:

```
from rexpro.utils import get_rexpro
sock_cls, conn_cls, pool_cls = get_rexpro()            # Returns the Synchronous classes
sock_cls, conn_cls, pool_cls = get_rexpro('sync')      # Returns the Synchronous classes
sock_cls, conn_cls, pool_cls = get_rexpro('gevent')    # Returns the Gevent classes
sock_cls, conn_cls, pool_cls = get_rexpro('eventlet')  # Returns the Eventlet classes
```

4.3 ChangeLog

Changes to the library are recorded here.

4.3.1 v0.4.5

- Long outstanding connection bug fixed by Estefan Ortiz! This is caused when something awry happens with rexster's rexpro socket either caused by titan or caused by network outtage

4.3.2 v0.4.4

- Remove unused graph features which polls every connection. - thanks to atmos

4.3.3 v0.4.3

- Loosen six version requirement

4.3.4 v0.4.2

- Connection Bugfix, connections that had thrown exceptions were sometimes unusable at this point, so fix logic to re-create the socket

4.3.5 v0.4.1

- python languange support via jython connectors like gremlin-python - thanks to Brian Corbin

4.3.6 v0.4.0

- Pooling support for Blueprints Wrapper and session management - thanks to aolieman

4.3.7 v0.3.3

- Fix Pooling issue: always return connections back to the pool by nvie

4.3.8 v0.3.2

- Support None in response

4.3.9 v0.3.1

- Transaction handling fix by nvie

4.3.10 v0.3.0

- Python 3 support finalized, backported for python2 compatibility

4.3.11 v0.2.2

- Better Python 3.3 and 3.4 testing

4.3.12 v0.2.1

- Bugfix for closing transaction - thanks to Ashald for fixing this

4.3.13 v0.2.0

- Robust Reconnect and session handling
- Allow Gevent, Eventlet and native python socket implementations
- **Concurrency options are optional to use the library**
  - To install with gevent: `pip install rexpro[gevent]`
  - To install with eventlet: `pip install rexpro[eventlet]`
- Documentation Improvements

4.3.14 v0.1.8

- Naive Reconnect

4.3.15 v0.1.7

- Documentation

4.3.16 v0.1.6

- Reconnect handling
- Testing against Celery * must use `-P gevent` flag, `eventlet` is known to immediately hang forever

4.3.17 v0.1.5

- Hotfix for connection pooling context manager

4.3.18 v0.1.4

- Connection Pooling and Green Thread Friendly sockets using the `gevent` library. * Using gevent.socket directly, no monkey patching.

4.3.19 v0.1.3

- Hotfix for new setup.py `install_requires`

4.3.20 v0.1.2

- Authentication Fixed
- Coverage Testing
- Fixed broken tests.

4.4 Contributing

4.4.1 License

rexpro is distributed under the Apache 2.0 License.

4.4.2 Issues

Issues should be opened through GitHub Issues; whenever possible, a pull request should be included.

4.4.3 Pull Requests

All pull requests should pass the test suite, which can be launched simply with:

```
$ make test
```

Note: Running test requires *nosetests*, *coverage*, *six*, *gevent*, *eventlet*, *redis* and *celery*. As well an available titan server.

In order to test coverage, please use:

```
$ pip install coverage
$ coverage erase; ./run_coverage.sh
```

4.4.4 Test Server

rexpro was designed for Titan, other graph databases that utilize Blueprints may be compatible, but further testing would be needed.

Currently Titan 0.4.4 is known to work and can be downloaded: Titan.

CHAPTER 5

Indices and tables

- *genindex*
- *modindex*
- *search*

r

A

B

C

D

E

F

G

I

J

L

M

O

P

Q

R

S

T