goblin Documentation

Release 0.2.0

David M. Brown

April 12, 2016

Contents

Goblin - Python Object Graph Mapper for the TinkerPop3 Gremlin Server

As Gremlin approached The TinkerPop, `mogwai` felt left behind and the closer he got, the more his world dissolved. He realized that all that he realized was just a realization and that all realized realizations are just as real as the need to evolve into something else - `goblin` was born...

CHAPTER 1

Releases

The latest release of `Goblin` is 0.2.0 *(coming soon)*.

CHAPTER 2

Requirements

`Goblin` uses `gremlinclient` to communicate with the Gremlin Server, and can use a variety of client/library combinations that work with different versions of Python. See the `gremlinclient` docs for more information.

Tornado

- Python 2.7+

aiohttp

- Python 3.4+

Tornado w/Asyncio

- Python 3.3+

Tornado w/Trollius

- Python 2.7

`Goblin` aims to provide full support for all TinkerPop3 enabled graph databases; however, it is currently only tested against Titan:db 1.x. This project is under active development, and early releases should be considered alpha as the API is not yet entirely stable.

CHAPTER 3

Installation

Install using pip:

```
$ pip install goblin
```

CHAPTER 4

Getting Started

A simple example using the default Tornado client with Python 2.7+:

```
from tornado import gen
from tornado.ioloop import IOLoop
from goblin import properties
from goblin import connection
from goblin.models import Vertex, Edge, V


class User(Vertex):
    name = properties.String()


class Follows(Edge):
    pass


@gen.coroutine
def go():
    goblin = yield User.create(name="Goblin")
    gremlin = yield User.create(name="Gremlin")
    gob_follows_grem = yield Follows.create(goblin, gremlin)
    # Find Gremlin's followers
    stream = yield V(gremlin).in_step().get()  # `in` is a reserved word
    followers = yield stream.read()
    return followers


connection.setup("ws://localhost:8182")
loop = IOLoop.current()
try:
  followers = loop.run_sync(go)
finally:
  loop.close()
  connection.tear_down()
```

4.1 Contributing

`Goblin` is under active development on Github, and contributions are welcome. More guidelines coming soon....

Contents:

4.1.1 Using Goblin

Goblin aims to provide an easy to use, intuitive API, without sacrificing the flexibility enabled by extensive user configuration. It is designed to 'just work' out of the box with most versions of Python (2.7+) using the tornado.ioloop.IOLoop and tornado.concurrent.Future class. But as the Python community moves forward with projects like asyncio, trollius, aiohttp (based on asyncio), and curio, developers want to be able to choose the async API, event loop and future implementation that works with their stack. To enable this, Goblin provides a pluggable client implementation that allows the user to choose the client, event loop, and future classes, provided that certain compatibility and API requirements are kept in mind.

This document aims to present an overview of the core functionality of Goblin in the simplest manner possible. In future versions, each of these sections will be expanded into a guide covering the complete functionality provided by Goblin. For a full reference, please see the API docs.

Setting up Goblin

In order to talk to the Gremlin Server, Goblin needs a Future, which can be any future implementation with a compatible API, and a Pool, which is typically inherits from gremlinclient.Pool. For a simple application, you can use *goblin.connection.setup()*, which allows you to define a Future and Pool as *goblin.connection* constants that will be used throughout Goblin:

```
>>> import asyncio
>>> from gremlinclient import aiohttp_client
>>> from goblin import connection

>>> connection.setup(
...     pool_class=aiohttp_client.Pool, future_class=asyncio.Future)
```

The function *goblin.connection.setup()* provides a wide variety of configuration options, please refer to the API docs for a complete description.

For more involved applications, it is often desirable to manage connection pool and futures explicitly. Goblin allows these parameters to be passed as keyword arguments to any caller of *goblin.connection.execute_query()*.

After using *goblin.connection.setup()*, it is important to call *goblin.connection.tear_down()* to clean up any remaining connections. Depending on the connection pool implementation, this method may or may not return a Future. Using gremlinclient.aiohttp_client.Pool:

```
>>> yield from connection.tear_down()
```

All of the following examples assume a gremlinclient.aiohttp_client.Pool and asyncio.Future

Creating models

The core functionality of Goblin lies in the models module, which allows you to define Python classes (*vertices*, *edges*, and *properties*) that are mapped to graph elements.

using the Gremlin query language:

```
>>> from goblin import models
>>> from goblin import properties

>>> class User(models.Vertex):
...     name = properties.String()
...     email = properties.Email()
...     url = properties.Url()
```

```
>>> class Follows(models.Edge):
        pass  # Edge can have properties just like Vertex
```

We can then use the method `create` to create nodes and edges:

```
>>> joe = yield from User.create(name='joe', email='joe@joe.com',
...                              url='http://joe.com')
>>> bob = yield from User.create(name='bob', email='bob@bob.com',
...                              url='http://bob.com')
>>> joe_follows_bob = yield from Follows.create(joe, bob)
```

This creates two vertices with the label "user" and one edge with the label "follows" in the graphdb.

Elements can be retrieved from the graphdb using class methods provided by the element implementations. These methods include `get` which allows you to retrieve an element by id, and `all`, which retrieves all elements with a label corresponding to the model class from the database:

```
>>> joe = yield from User.get(joe.id)
>>> users = yield from User.all()
```

Instances of graph elements (Vertices and Edges) provide methods that allow you to delete and update properties.

```
>>> josep = yield from joe.save(name='Josep')
>>> yield from josep.delete()
```

Graph element instances also provide an API that allows you to access and modify neighbor elements, but this API is under review and may be deprecated in favor of the vertex centric query API and the proposed edge centric query API.

Using the `Relationship` class

In an effort to provide a more convenient API, `Goblin` provides the *`Relationship`* class, which allows you to explicitly define relationships between vertex classes:

```
>>> from goblin import models
>>> from goblin import properties
>>> from goblin import relationships

>>> class WorksIn(models.Edge):
        pass

>>> class Department(models.Vertex):
...     name = properties.String()

>>> class Employee(models.Vertex)
...     name = properties.String()
...     email = properties.Email()
...     department = relationships.Relationship(WorksIn, Department)
```

You can then use the `department` relationship to easily create edges of type *WorksIn* and vertices of type *Department*:

```
>>> joe = Employee.create(name='joe', email="joe@joe.com")
>>> joe_works_in, r_and_d = yield from joe.department.create(
...     vertex_params={'name': 'R&D'})
```

The *`Relationship`* class provides several other methods for convenience as well. For a full reference, please see the API docs

The Vertex centric query API

To emulate Gremlin style traversals, `Goblin` provides the class *V*, which provides an interface for step based graph traversals using method chaining. Unlike Gremlin, the class *V* requires that the user pass a vertex or vertex id as a starting point for the traversal. For example:

```
>>> from goblin.models import V
>>> dep = yield from V(joe).out_step().get()
>>> r_and_d = yield from V(joe).\
...     out_step().\
...     has(Department.get_property_by_name('name'), 'R&D').\
...     get()
```

There are three things to note in the above example:

1. The Gremlin steps *in* and *out* have been renamed as *in_step* and *out_step* due to the fact that *in* is a reserved word in Python
2. All traversal must end with *get*.
3. The *has* step requires that you use `get_property_by_name()` method to retrieve the correct property key.

Furthermore, it should be noted that the camel case used in Gremlin steps has been replaced with the underscores more commonly used with Python methods: *inV -> in_v*.

For a full list of steps, please see the API docs

Coming soon, detailed guides...

4.1.2 Choosing a Websocket Client

The choice of client is entirely up to the user. For more information on available client implementations, please see the `gremlinclient` API documentation.

Current compatible client implementations included with `gremlinclient`:

- `Tornado`
- `aiohttp`

4.1.3 Schema Management Utilities

Coming soon...

4.1.4 Integrating `Goblin` with Asynchronous Web Frameworks

Coming soon:

Integrating Goblin with Tornado

Integrating Goblin with aiohttp

Integrating Goblin with Pulsar

4.1.5 Goblin API

Subpackages

goblin.gremlin package

Submodules

goblin.gremlin.base module

class goblin.gremlin.base.**BaseGremlinMethod**(*path=None*, *method_name=None*, *classmethod=False*, *property=False*, *defaults=None*, *transaction=True*, *imports=None*)

Bases: object

Maps a function in a groovy file to a method on a python class

configure_method(*klass*, *attr_name*, *gremlin_path*)

Sets up the methods internals

Parameters

- **klass** (*object*) – The class object this function is being added to
- **attr_name** (*str*) – The attribute name this function will be added as
- **gremlin_path** (*str*) – The path to the gremlin file containing method

transform_params_to_database(*params*)

Takes a dictionary of parameters and recursively translates them into parameters appropriate for sending over Rexpro.

Parameters **params** (*dict*) – The parameters to be sent to the function

Return type dict

class goblin.gremlin.base.**GremlinMethod**(*path=None*, *method_name=None*, *classmethod=False*, *property=False*, *defaults=None*, *transaction=True*, *imports=None*)

Bases: *goblin.gremlin.base.BaseGremlinMethod*

Gremlin method that returns a graph element

class goblin.gremlin.base.**GremlinTable**(*path=None*, *method_name=None*, *classmethod=False*, *property=False*, *defaults=None*, *transaction=True*, *imports=None*)

Bases: *goblin.gremlin.base.GremlinMethod*

Gremlin method that returns a table as its result

class goblin.gremlin.base.**GremlinValue**(*path=None*, *method_name=None*, *classmethod=False*, *property=False*, *defaults=None*, *transaction=True*, *imports=None*)

Bases: *goblin.gremlin.base.GremlinMethod*

Gremlin Method that returns one value

goblin.gremlin.base.**groovy_import**(*extra_import*)

goblin.gremlin.groovy module

goblin.gremlin.groovy.**GroovyFileDef**
alias of GroovyFileDefinition

class goblin.gremlin.groovy.**GroovyFunction**(*name*, *args*, *body*, *defn*)
Bases: tuple

args
Alias for field number 1

body
Alias for field number 2

defn
Alias for field number 3

name
Alias for field number 0

class goblin.gremlin.groovy.**GroovyFunctionParser**
Bases: object

Given a string containing a single function definition this class will parse the function definition and return information regarding it.

FuncDefn = {{{{{“def” Re:(‘[A-Za-z_]\\w*’)} “(“} Re:(‘[A-Za-z_]\\w*’) [, Re:(‘[A-Za-z_]\\w*’)]...} ”)”} “{“}

FuncName = Re:(‘[A-Za-z_]\\w*’)

KeywordDef = “def”

VarName = Re:(‘[A-Za-z_]\\w*’)

classmethod parse(*data*)
Parse the given function definition and return information regarding the contained definition.

Parameters **data** (*str | basestring*) – The function definition in a string

Return type dict

class goblin.gremlin.groovy.**GroovyImport**(*comment_list*, *import_strings*, *import_list*)
Bases: tuple

comment_list
Alias for field number 0

import_list
Alias for field number 2

import_strings
Alias for field number 1

class goblin.gremlin.groovy.**GroovyImportParser**
Bases: object

Given a string containing a single import definition this class will parse the import definition and return information regarding it.

CommentVar = comment

ImportDef = Suppress:(“import”)

ImportDefn = {{{Suppress:(“import”) Re:(‘[A-Za-z_.*]*’) [. Re:(‘[A-Za-z_.*]*’)]...} Suppress:(”;”)} [{Suppress:(“//”)

ImportVarName = Re:('[A-Za-z_.*]*')

OptionalSpace = [" "]

classmethod parse(*data*)

Parse the given import and return information regarding the contained import statement.

Parameters **data** (*str | basestring*) – The import statement in a string

Return type dict

goblin.gremlin.groovy.**parse**(*filename*)

Parse Groovy code in the given file and return a list of information about each function necessary for usage in queries to database.

Parameters **filename** (*str*) – The file containing groovy code.

Return type list

goblin.gremlin.table module

class goblin.gremlin.table.**Table**(*gremlin_result*)

Bases: object

A table accepts the results of a GremlinTable in it's constructor. It can be iterated over like a normal list, but within the rows the dictionaries are accessible via .notation

For example:

returns a table of people & my friend edge to them # the edge contains my nickname for that person friends = goblin.gremlin.GremlinTable()

def get_friends_and_my_nickname(self): result = self.friends() for i in result:

print "{}:{}".format(i.friend_edge.nickname, i.person.name)

next()

class goblin.gremlin.table.**Row**(*data*)

Bases: object

A row represent a table row, from which it's columns can be accessed like a tuple or dict. Rows are read-only and accept elements or dicts with as initializers as a result of a GremlinTable query. Also the . getattr notation can be used to access elements

Example: row = Row({'person': Friend.create(....), 'myval': 3}) print "{}:{} - {}".format(row.friend_edge.nickname, row.person.name, row.myval)

items()

iteritems()

keys()

next()

values()

goblin.models package

Submodules

goblin.models.edge module

class goblin.models.edge.**Edge**(*outV*, *inV*, ***values*)

Bases: *goblin.models.element.Element*

Base class for all edges.

FACTORY_CLASS = None

classmethod all(*ids*, *as_dict=False*, **args*, ***kwargs*)

Load all edges with the given edge_ids from the graph. By default this will return a list of edges but if as_dict is True then it will return a dictionary containing edge_ids as keys and edges found as values.

Parameters

- **ids** (*list*) – A list of titan IDs
- **as_dict** (*boolean*) – Toggle whether to return a dictionary or list

Return type dict | list

classmethod create(*outV*, *inV*, *label=None*, **args*, ***kwargs*)

Create a new edge of the current type coming out of vertex outV and going into vertex inV with the given properties.

Parameters

- **outV** (Vertex) – The vertex the edge is coming out of
- **inV** (Vertex) – The vertex the edge is going into

delete(***kwargs*)

Delete the current edge from the graph.

classmethod find_by_value(*field*, *value*, *as_dict=False*, ***kwargs*)

Returns edges that match the given field/value pair.

Parameters

- **field** (*str*) – The field to search
- **value** (*str*) – The value of the field
- **as_dict** (*boolean*) – Return results as a dictionary

Return type [goblin.models.Edge]

classmethod get(*id*, **args*, ***kwargs*)

Look up edge by titan assigned ID. Raises a DoesNotExist exception if an edge with the given edge id was not found. Raises a MultipleObjectsReturned exception if the edge_id corresponds to more than one edge in the graph.

Parameters **id** (*str | basestring*) – The titan assigned ID

Return type goblin.models.Edge

classmethod get_between(*outV*, *inV*, *page_num=None*, *per_page=None*)

Return all the edges with a given label between two vertices.

Parameters

- **outV** (Vertex) – The vertex the edge comes out of.
- **inV** (Vertex) – The vertex the edge goes into.
- **page_num** (*int*) – The page number of the results
- **per_page** (*int*) – The number of results per page

Return type list

classmethod get_label()
Returns the label for this edge.

Return type str

gremlin_path = 'edge.groovy'

inV(**args*, ***kwargs*)
Return the vertex that this edge goes into.

Return type *Vertex*

label = None

outV(**args*, ***kwargs*)
Return the vertex that this edge goes into.

Return type *Vertex*

save(**args*, ***kwargs*)
Save this edge to the graph database.

validate()
Perform validation of this edge raising a ValidationError if any problems are encountered.

class goblin.models.edge.**EdgeMetaClass**
Bases: *goblin.models.element.ElementMetaClass*

Metaclass for edges.

goblin.models.element module

class goblin.models.element.**BaseElement**(***values*)
Bases: object

The base model class, don't inherit from this, inherit from Model, defined below

exception DoesNotExist
Bases: *goblin.exceptions.GoblinException*

Object not found in database

BaseElement.**FACTORY_CLASS = None**

exception BaseElement.**MultipleObjectsReturned**
Bases: *goblin.exceptions.GoblinException*

Multiple objects returned on unique key lookup

exception BaseElement.**WrongElementType**
Bases: *goblin.exceptions.GoblinException*

Unique lookup with key corresponding to vertex of different type

BaseElement.**as_dict()**
Returns a map of column names to cleaned values

Return type dict

BaseElement.**as_save_params()**
Returns a map of property names to cleaned values containing only the properties which should be persisted on save.

Return type dict

classmethod BaseElement.**create**(**args*, ***kwargs*)
Create a new element with the given information.

classmethod BaseElement.**get_property_by_name**(*key*)
Get's the db_field_name of a property by key

Parameters **key** (*basestring | str*) – attribute of the model

Return type basestring | str | None

BaseElement.**id**

BaseElement.**items**()

BaseElement.**keys**()

BaseElement.**label**

BaseElement.**pre_save**()
Pre-save hook which is run before saving an element

BaseElement.**pre_update**(***values*)
Override this to perform pre-update validation

BaseElement.**reload**(**args*, ***kwargs*)
Reload the given element from the database.

BaseElement.**save**()
Base class save method. Performs basic validation and error handling.

classmethod BaseElement.**translate_db_fields**(*data*)
Translates field names from the database into field names used in our model this is for cases where we're saving a field under a different name than it's model property

Parameters **data** – dict

Return type dict

BaseElement.**update**(***values*)
performs an update of this element with the given values and returns the saved object

BaseElement.**validate**()
Cleans and validates the field values

BaseElement.**validate_field**(*field_name*, *val*)
Perform the validations associated with the field with the given name on the value passed.

Parameters

- **field_name** (*str*) – The name of property whose validations will be run
- **val** (*mixed*) – The value to be validated

BaseElement.**values**()

class goblin.models.element.**Element**(***values*)
Bases: *goblin.models.element.BaseElement*

classmethod **deserialize**(*data*)
Deserializes rexpro response into vertex or edge objects

gremlin_path = None

class goblin.models.element.**ElementMetaClass**
Bases: type

Metaclass for all graph elements

goblin.models.query module

class goblin.models.query.**V**(*vertex*)

Bases: object

All query operations return a new query object, which currently deviates from blueprints. The blueprints query object modifies and returns the same object This method seems more flexible, and consistent w/ the rest of Gremlin.

both(**labels*)

both_e(**labels*)

both_v()

count(**args*, ***kwargs*)

Returns number of matching vertices

Return type int

get(*deserialize=True*, **args*, ***kwargs*)

has(*key*, *value*, *compare='eq'*)

Parameters

- **key** (*str | goblin.properties.GraphProperty*) – key to lookup
- **value** (*str, float, int*) – value to compare
- **compare** (*str*) – comparison keyword

Return type Query

has_id(**ids*)

has_label(**labels*)

in_e(**labels*)

in_step(**labels*)

in_v()

limit(*limit*)

other_v()

out_e(**labels*)

out_step(**labels*)

out_v()

goblin.models.vertex module

class goblin.models.vertex.**EnumVertexBaseMeta**

Bases: *goblin.models.vertex.VertexMetaClass*

This metaclass allows you to access MyVertexModel as if it were an enum. Ex. MyVertexModel.FOO

The values are cached in a dictionary. This is useful if the number of MyVertexModels is small, however it it grows too large, you should be doing it a different way.

This looks for a special (optional) function named *enum_generator* in your model and calls that to generate the ENUM for the model.

There is an additional optional model attribute that can be set *__enum_id_only__* (defaults to True) which dictates whether or not just the Vertex ID is stored, or the whole Vertex in cache.

enums = None

class goblin.models.vertex.**Vertex**(**values*)

Bases: *goblin.models.element.Element*

The Vertex model base class.

The element type is auto-generated from the subclass name, but can optionally be set manually

FACTORY_CLASS = None

classmethod all(*ids=[]*, *as_dict=False*, *match_length=True*, **args*, ***kwargs*)

Load all vertices with the given ids from the graph. By default this will return a list of vertices but if as_dict is True then it will return a dictionary containing ids as keys and vertices found as values.

Parameters

- **ids** (*list*) – A list of titan ids
- **as_dict** (*boolean*) – Toggle whether to return a dictionary or list

Return type dict | list

bothE(**labels*, ***kwargs*)

Return a list of edges both incoming and outgoing from this vertex.

Parameters

- **label** (*str or BaseEdge or None*) – The edge label to be traversed (optional)
- **limit** (*int or None*) – The number of the page to start returning results at
- **offset** (*int or None*) – The maximum number of results to return
- **types** (*list*) – A list of allowed element types

bothV(**labels*, ***kwargs*)

Return a list of vertices both incoming and outgoing from this vertex.

Parameters

- **label** (*str or BaseEdge or None*) – The edge label to be traversed (optional)
- **limit** (*int or None*) – The number of the page to start returning results at
- **offset** (*int or None*) – The maximum number of results to return
- **types** (*list*) – A list of allowed element types

delete(***kwargs*)

Delete the current vertex from the graph.

delete_inE(**labels*, ***kwargs*)

Delete all incoming edges with the given label.

delete_inV(**labels*, ***kwargs*)

Delete all incoming vertices connected with edges with the given label.

delete_outE(**labels*, ***kwargs*)

Delete all outgoing edges with the given label.

delete_outV(**labels*, ***kwargs*)

Delete all outgoing vertices connected with edges with the given label.

classmethod **find_by_value**(*field*, *value*, *as_dict=False*)
Returns vertices that match the given field/value pair.

Parameters

- **field** (*str*) – The field to search
- **value** (*str*) – The value of the field
- **as_dict** (*boolean*) – Return results as a dictionary

Return type [goblin.models.Vertex]

classmethod **get**(*id*, **args*, ***kwargs*)
Look up vertex by its ID. Raises a DoesNotExist exception if a vertex with the given vid was not found. Raises a MultipleObjectsReturned exception if the vid corresponds to more than one vertex in the graph.

Parameters **id** (*str*) – The ID of the vertex

Return type goblin.models.Vertex

classmethod **get_label**()
Returns the element type for this vertex.

@returns: str

gremlin_path = 'vertex.groovy'

inE(**labels*, ***kwargs*)
Return a list of edges with the given label coming into this vertex.

Parameters

- **label** (*str or BaseEdge*) – The edge label to be traversed
- **limit** (*int or None*) – The number of the page to start returning results at
- **offset** (*int or None*) – The maximum number of results to return
- **types** (*list*) – A list of allowed element types

inV(**labels*, ***kwargs*)
Return a list of vertices reached by traversing the incoming edge with the given label.

Parameters

- **label** (*str or BaseEdge*) – The edge label to be traversed
- **limit** (*int or None*) – The number of the page to start returning results at
- **offset** (*int or None*) – The maximum number of results to return
- **types** (*list*) – A list of allowed element types

label = None

outE(**labels*, ***kwargs*)
Return a list of edges with the given label going out of this vertex.

Parameters

- **label** (*str or BaseEdge*) – The edge label to be traversed
- **limit** (*int or None*) – The number of the page to start returning results at
- **offset** (*int or None*) – The maximum number of results to return
- **types** (*list*) – A list of allowed element types

outV(**labels*, ***kwargs*)
Return a list of vertices reached by traversing the outgoing edge with the given label.

Parameters

- **labels** (*str or BaseEdge*) – pass in the labels to follow in as positional arguments
- **limit** (*int or None*) – The number of the page to start returning results at
- **offset** (*int or None*) – The maximum number of results to return
- **types** (*list*) – A list of allowed element types

query()

save(**args*, ***kwargs*)
Save the current vertex using the configured save strategy, the default save strategy is to re-save all fields every time the object is saved.

class goblin.models.vertex.**VertexMetaClass**
Bases: *goblin.models.element.ElementMetaClass*

Metaclass for vertices.

goblin.properties package

Submodules

goblin.properties.base module

class goblin.properties.base.**BaseValueManager**(*graph_property*, *value*, *strategy=<class 'goblin.properties.strategy.SaveAlways'>*)
Bases: object

Value managers are used to manage values pulled from the database and track state changes.

These are useful for save strategies.

changed
Indicates whether or not this value has changed.

Return type bool

deleted
Indicates whether or not this value has been deleted.

Return type bool

delval()
Delete a given value

get_property()
Returns a value-managed property attributes

Return type property

getval()
Return the current value.

previous_value

setval(*val*)

Updates the current value.

param val The new value

class goblin.properties.base.**GraphProperty**(*description=None*, *primary_key=False*, *index=False*, *index_ext=None*, *db_field=None*, *choices=None*, *default=None*, *required=False*, *save_strategy=<class 'goblin.properties.strategy.SaveAlways'>*, *unique=None*, *db_field_prefix=''*)

Bases: object

Base class for graph property types

can_delete

data_type = 'Object'

db_field_name
Returns the name of the goblin name of this graph property

Return type basestring | str

get_default()
Returns the default value for this graph property if one is available.

Return type Object | None

get_save_strategy()
Returns the save strategy attached to this graph property.

Return type Callable

classmethod get_value_from_choices(*value*, *choices*)
Returns the key for the choices tuple of tuples

Note if you are using classes, they must implement the __in__ and __eq__ for the logical comparison.

Parameters **value** (*Object*) – The raw value to test if it exists in the valid choices. Could be the key or the value in the dict

Return type Object

has_db_field_prefix
Determines if a field prefix has already been defined.

has_default
Indicates whether or not this graph property has a default value.

Return type bool

instance_counter = 0

set_db_field_prefix(*prefix*, *override=False*)
Sets the graph property name prefix during document class construction.

set_property_name(*name*)
Sets the graph property name during document class construction.

This value will be ignored if db_field is set in __init__

Parameters **name** (*str*) – The name of this graph property

should_save(*first_save=False*)
Indicates whether or not the property should be saved based on it's save strategy.

Return type bool

to_database(*value*)
Converts python value into database value

Return type Object

to_python(*value*)
Converts data from the database into python values raises a ValidationError if the value can't be converted

Return type Object

validate(*value*)
Returns a cleaned and validated value. Raises a ValidationError if there's a problem

Return type Object

validator = <goblin.properties.validators.BaseValidator object>

value_manager
alias of *BaseValueManager*

goblin.properties.properties module

class goblin.properties.properties.**Boolean**(*description=None*, *primary_key=False*, *index=False*, *index_ext=None*, *db_field=None*, *choices=None*, *default=None*, *required=False*, *save_strategy=<class 'goblin.properties.strategy.SaveAlways'>*, *unique=None*, *db_field_prefix=''*)
Bases: *goblin.properties.base.GraphProperty*

Boolean Data property type

data_type = 'Boolean'

to_database(*value*)

to_python(*value*)

validator = <goblin.properties.validators.BooleanValidator object>

class goblin.properties.properties.**DateTime**(*strict=True*, ***kwargs*)
Bases: *goblin.properties.base.GraphProperty*

UTC DateTime Data property type

data_type = 'Double'

to_database(*value*)

to_python(*value*)

validator = <goblin.properties.validators.DateTimeUTCValidator object>

class goblin.properties.properties.**DateTimeNaive**(*strict=True*, ***kwargs*)
Bases: *goblin.properties.base.GraphProperty*

DateTime Data property type

data_type = 'Double'

to_database(*value*)

to_python(*value*)

validator = <goblin.properties.validators.DateTimeValidator object>

class goblin.properties.properties.**Decimal**(*description=None*, *primary_key=False*, *index=False*, *index_ext=None*, *db_field=None*, *choices=None*, *default=None*, *required=False*, *save_strategy=<class 'goblin.properties.strategy.SaveAlways'>*, *unique=None*, *db_field_prefix=''*)

Bases: *goblin.properties.base.GraphProperty*

Decimal Data property type

to_database(*value*)

to_python(*value*)

validator = <goblin.properties.validators.DecimalValidator object>

class goblin.properties.properties.**Dictionary**(*description=None*, *primary_key=False*, *index=False*, *index_ext=None*, *db_field=None*, *choices=None*, *default=None*, *required=False*, *save_strategy=<class 'goblin.properties.strategy.SaveAlways'>*, *unique=None*, *db_field_prefix=''*)

Bases: *goblin.properties.base.GraphProperty*

Dictionary Data property type

data_type = 'HashMap'

validator = <goblin.properties.validators.DictValidator object>

class goblin.properties.properties.**Double**(***kwargs*)

Bases: *goblin.properties.base.GraphProperty*

Double Data property type

data_type = 'Double'

to_database(*value*)

to_python(*value*)

validator = <goblin.properties.validators.FloatValidator object>

class goblin.properties.properties.**Email**(**args*, ***kwargs*)

Bases: *goblin.properties.base.GraphProperty*

Email Data property type

data_type = 'String'

validate(*value*)

validator = <goblin.properties.validators.EmailValidator object>

class goblin.properties.properties.**Float**(***kwargs*)

Bases: *goblin.properties.properties.Double*

Float class for backwards compatability / if you really want to

class goblin.properties.properties.**IPV4**(**args*, ***kwargs*)

Bases: *goblin.properties.base.GraphProperty*

IPv4 Data property type

data_type = 'String'

validate(*value*)

validator = <goblin.properties.validators.RegexValidator object>

class goblin.properties.properties.**IPV6**(**args*, ***kwargs*)
Bases: *goblin.properties.base.GraphProperty*

IPv6 Data property type

data_type = 'String'

validate(*value*)

validator = <goblin.properties.validators.RegexValidator object>

class goblin.properties.properties.**IPV6WithV4**(**args*, ***kwargs*)
Bases: *goblin.properties.base.GraphProperty*

IPv6 with Mapped/Translated/Embedded IPv4 Data property type

data_type = 'String'

validate(*value*)

validator = <goblin.properties.validators.RegexValidator object>

class goblin.properties.properties.**Integer**(*description=None*, *primary_key=False*, *index=False*, *index_ext=None*, *db_field=None*, *choices=None*, *default=None*, *required=False*, *save_strategy=<class 'goblin.properties.strategy.SaveAlways'>*, *unique=None*, *db_field_prefix=''*)
Bases: *goblin.properties.base.GraphProperty*

Integer Data property type

data_type = 'Integer'

to_database(*value*)

to_python(*value*)

validator = <goblin.properties.validators.LongValidator object>

class goblin.properties.properties.**List**(*description=None*, *primary_key=False*, *index=False*, *index_ext=None*, *db_field=None*, *choices=None*, *default=None*, *required=False*, *save_strategy=<class 'goblin.properties.strategy.SaveAlways'>*, *unique=None*, *db_field_prefix=''*)
Bases: *goblin.properties.base.GraphProperty*

List Data property type

data_type = 'ArrayList'

validator = <goblin.properties.validators.ListValidator object>

class goblin.properties.properties.**Long**(*description=None*, *primary_key=False*, *index=False*, *index_ext=None*, *db_field=None*, *choices=None*, *default=None*, *required=False*, *save_strategy=<class 'goblin.properties.strategy.SaveAlways'>*, *unique=None*, *db_field_prefix=''*)
Bases: *goblin.properties.base.GraphProperty*

Long Data property type

data_type = 'Long'

to_database(*value*)

to_python(*value*)

validator = <goblin.properties.validators.LongValidator object>

class goblin.properties.properties.**PositiveInteger**(*description=None*, *primary_key=False*, *index=False*, *index_ext=None*, *db_field=None*, *choices=None*, *default=None*, *required=False*, *save_strategy=<class 'goblin.properties.strategy.SaveAlways'>*, *unique=None*, *db_field_prefix=''*)

Bases: *goblin.properties.properties.Integer*

Positive Integer Data property type

data_type = 'Integer'

to_database(*value*)

to_python(*value*)

validator = <goblin.properties.validators.PositiveIntegerValidator object>

class goblin.properties.properties.**PositiveLong**(*description=None*, *primary_key=False*, *index=False*, *index_ext=None*, *db_field=None*, *choices=None*, *default=None*, *required=False*, *save_strategy=<class 'goblin.properties.strategy.SaveAlways'>*, *unique=None*, *db_field_prefix=''*)

Bases: *goblin.properties.properties.Long*

Positive Long Data property type

data_type = 'Long'

to_database(*value*)

to_python(*value*)

validator = <goblin.properties.validators.PositiveIntegerValidator object>

class goblin.properties.properties.**Short**(*description=None*, *primary_key=False*, *index=False*, *index_ext=None*, *db_field=None*, *choices=None*, *default=None*, *required=False*, *save_strategy=<class 'goblin.properties.strategy.SaveAlways'>*, *unique=None*, *db_field_prefix=''*)

Bases: *goblin.properties.base.GraphProperty*

Short Data property type

data_type = 'Short'

to_database(*value*)

to_python(*value*)

validator = <goblin.properties.validators.IntegerValidator object>

class goblin.properties.properties.**Slug**(**args*, ***kwargs*)

Bases: *goblin.properties.base.GraphProperty*

Slug Data property type

data_type = 'String'

validate(*value*)

validator = <goblin.properties.validators.RegexValidator object>

class goblin.properties.properties.**String**(**args*, ***kwargs*)
Bases: *goblin.properties.base.GraphProperty*

String/CharField property

data_type = 'String'

to_python(*value*)

validate(*value*)

validator = <goblin.properties.validators.StringValidator object>

goblin.properties.properties.**Text**
alias of *String*

class goblin.properties.properties.**URL**(**args*, ***kwargs*)
Bases: *goblin.properties.base.GraphProperty*

URL Data property type

data_type = 'String'

validate(*value*)

validator = <goblin.properties.validators.URLValidator object>

class goblin.properties.properties.**UUID**(*default=<function UUID.<lambda>>*, ***kwargs*)
Bases: *goblin.properties.base.GraphProperty*

Universally Unique Identifier (UUID) type - UUID4 by default

data_type = 'String'

to_database(*value*)

to_python(*value*)

validator = <goblin.properties.validators.RegexValidator object>

goblin.properties.strategy module

class goblin.properties.strategy.**SaveAlways**
Bases: *goblin.properties.strategy.Strategy*

Save this value every time the corresponding model is saved.

classmethod condition(*previous_value*, *value*, *has_changed=False*, *first_save=False*, *graph_property=None*)
Save this value every time the corresponding model is saved.

Return type bool

class goblin.properties.strategy.**SaveOnChange**
Bases: *goblin.properties.strategy.Strategy*

Only save this value if it has changed.

classmethod condition(*previous_value*, *value*, *has_changed=False*, *first_save=False*, *graph_property=None*)
Always save this value if it has changed

Return type bool

class goblin.properties.strategy.**SaveOnDecrease**
Bases: *goblin.properties.strategy.Strategy*

Save this value only if it is decreasing

classmethod condition(*previous_value*, *value*, *has_changed=False*, *first_save=False*, *graph_property=None*)
Only save this value if it is decreasing

Return type bool

class goblin.properties.strategy.**SaveOnIncrease**
Bases: *goblin.properties.strategy.Strategy*

Save this value only if it is increasing

classmethod condition(*previous_value*, *value*, *has_changed=False*, *first_save=False*, *graph_property=None*)
Only save this value if it is increasing

Return type bool

class goblin.properties.strategy.**SaveOnce**
Bases: *goblin.properties.strategy.Strategy*

Only save this value once. If it changes throw an exception.

classmethod condition(*previous_value*, *value*, *has_changed=False*, *first_save=False*, *graph_property=None*)
Always save this value if it has changed

Raises SaveStrategyException

Return type bool

class goblin.properties.strategy.**Strategy**
Bases: object

Saving strategies for goblin. These are used to indicate when a property should be saved after the initial vertex/edge creation.

classmethod condition(*previous_value*, *value*, *has_changed=False*, *first_save=False*, *graph_property=None*)
Default save strategy condition

Raises NotImplementedError

goblin.properties.validators module

class goblin.properties.validators.**BaseValidator**(*message=None*, *code=None*)
Bases: object

code = 'invalid'

message = 'Enter a valid value.'

class goblin.properties.validators.**BooleanValidator**(*message=None*, *code=None*)
Bases: *goblin.properties.validators.BaseValidator*

message = 'Enter a valid Boolean.'

class goblin.properties.validators.**DateTimeUTCValidator**(*message=None*, *code=None*)
Bases: *goblin.properties.validators.BaseValidator*

message = 'Not a valid UTC DateTime: {}'

class goblin.properties.validators.**DateTimeValidator**(*message=None*, *code=None*)
Bases: *goblin.properties.validators.BaseValidator*

message = 'Not a valid DateTime: {}'

class goblin.properties.validators.**DecimalValidator**(*message=None*, *code=None*)
Bases: *goblin.properties.validators.NumericValidator*

data_types = (<class 'float'>, <class 'decimal.Decimal'>)

class goblin.properties.validators.**DictValidator**(*message=None*, *code=None*)
Bases: *goblin.properties.validators.BaseValidator*

message = 'Enter a valid dict'

class goblin.properties.validators.**EmailValidator**(*regex=None*, *message=None*, *code=None*)
Bases: *goblin.properties.validators.RegexValidator*

code = 'invalid'

message = 'Enter a valid email address: {}'

regex = re.compile('(^[-!#$%&\'*+/=?^_`{}|~0-9A-Z]+(\\.[-!#$%&\'*+/=?^_`{}|~0-9A-Z]+)*|^"([\\001-\\010\\013\\014\\01

class goblin.properties.validators.**FloatValidator**(*message=None*, *code=None*)
Bases: *goblin.properties.validators.NumericValidator*

data_types = (<class 'float'>,)

class goblin.properties.validators.**IntegerValidator**(*message=None*, *code=None*)
Bases: *goblin.properties.validators.NumericValidator*

data_types = (<class 'int'>,)

class goblin.properties.validators.**ListValidator**(*message=None*, *code=None*)
Bases: *goblin.properties.validators.BaseValidator*

data_types = (<class 'tuple'>, <class 'list'>)

message = 'Enter a valid list'

class goblin.properties.validators.**LongValidator**(*message=None*, *code=None*)
Bases: *goblin.properties.validators.NumericValidator*

data_types = (<class 'int'>,)

class goblin.properties.validators.**NumericValidator**(*message=None*, *code=None*)
Bases: *goblin.properties.validators.BaseValidator*

data_types = (<class 'float'>, <class 'int'>, <class 'decimal.Decimal'>)

message = 'Enter a valid number.'

class goblin.properties.validators.**PositiveIntegerValidator**(*message=None*, *code=None*)
Bases: *goblin.properties.validators.NumericValidator*

data_types = (<class 'int'>,)

class goblin.properties.validators.**RegexValidator**(*regex=None*, *message=None*, *code=None*)
Bases: *goblin.properties.validators.BaseValidator*

regex = ''

class goblin.properties.validators.**StringValidator**(*message=None*, *code=None*)
Bases: *goblin.properties.validators.BaseValidator*

data_type = (<class 'str'>,)

message = 'Enter a valid string: {}'

class goblin.properties.validators.**URLValidator**(*regex=None*, *message=None*, *code=None*)

Bases: *goblin.properties.validators.RegexValidator*

code = 'invalid'

message = 'Enter a valid URL address: {}'

regex = re.compile('^(?:http|ftp)s?://(?:(?:[A-Z0-9](?:[A-Z0-9-]{0,61}[A-Z0-9])?\\.)+(?:[A-Z]{2,6}\\.?|[A-Z0-9-]{2,}\\.?)|lo

goblin.relationships package

Submodules

goblin.relationships.base module

class goblin.relationships.base.**Relationship**(*edge_class*, *vertex_class*, *direction='both'*, *strict=True*, *gremlin_path=None*, *vertex_callback=None*, *edge_callback=None*, *query_callback=None*, *create_callback=None*)

Bases: object

Define incoming and outgoing relationships that exist. Also enforce schema IN, OUT and BOTH directions

Warn if queries return schema violations.

allowed(*edge_type*, *vertex_type*)

Check whether or not the allowed Edge and Vertex type are compatible with the schema defined

Parameters

- **edge_type** – Edge Class
- **vertex_type** – Vertex Class

Type goblin.models.Edge

Type goblin.models.Vertex

Return type bool

create(*edge_params={}*, *vertex_params={}*, *edge_type=None*, *vertex_type=None*, *callback=None*, ***kwargs*)

Creates a Relationship defined by the schema

Parameters

- **edge_params** (*dict*) – (Optional) Parameters passed to the instantiation method of the Edge
- **vertex_params** (*dict*) – (Optional) Parameters passed to the instantiation method
- **edge_type** (*goblin.models.Vertex | None*) – (Optional) Edge class type, otherwise it defaults to the first Edge type known
- **edge_type** – (Optional) Vertex class type, otherwise it defaults to the first Vertex type known
- **callback** (*method*) – (Optional) Callback function to handle results

Return type tuple(goblin.models.Edge, goblin.models.Vertex) | Object

edges(*limit=None*, *offset=None*, *callback=None*, ***kwargs*)
Query and return all Edges attached to the current Vertex

TODO: fix this, the instance method isn't properly setup :param limit: Limit the number of returned results :type limit: int | long :param offset: Query offset of the number of paginated results :type offset: int | long :param callback: (Optional) Callback function to handle results :type callback: method :rtype: List[goblin.models.Edge] | Object

query(*edge_types=None*, *callback=None*)
Generic Query method for quick access

Parameters

- **edge_types** (*List[goblin.models.Edge] | None*) – List of Edge classes to query against
- **callback** (*method*) – (Optional) Callback function to handle results

Return type goblin.models.query.Query | Object

vertices(*limit=None*, *offset=None*, *callback=None*, ***kwargs*)
Query and return all Vertices attached to the current Vertex

TODO: fix this, the instance method isn't properly setup :param limit: Limit the number of returned results :type limit: int | long :param offset: Query offset of the number of paginated results :type offset: int | long :param callback: (Optional) Callback function to handle results :type callback: method :rtype: List[goblin.models.Vertex] | Object

goblin.relationships.base.**requires_vertex**(*method*)

Submodules

goblin.connection module

goblin.connection.**execute_query**(*query*, *bindings=None*, *pool=None*, *future_class=None*, *graph_name=None*, *traversal_source=None*, *username=''*, *password=''*, *handler=None*, *request_id=None*, **args*, ***kwargs*)
Execute a raw Gremlin query with the given parameters passed in.

Parameters

- **query** (*str*) – The Gremlin query to be executed
- **bindings** (*dict*) – Bindings for the Gremlin query
- **pool** (*gremlinclient.pool.Pool*) – Pool that provides connection used in query
- **graph_name** (*str*) – graph name as defined in server configuration. Defaults to "graph"
- **traversal_source** (*str*) – traversal source name as defined in the server configuration. Defaults to "g"
- **username** (*str*) – username as defined in the Tinkerpop credentials graph.
- **password** (*str*) – password for username as definined in the Tinkerpop credentials graph
- **handler** (*func*) – Handles preprocessing of query results

Returns Future

goblin.connection.**generate_spec**()

goblin.connection.**get_future**(*kwargs*)

goblin.connection.**pop_execute_query_kwargs**(*keyword_arguments*)
pop the optional execute query arguments from arbitrary kwargs; return non-None query kwargs in a dict

goblin.connection.**setup**(*url*, *pool_class=None*, *graph_name='graph'*, *traversal_source='g'*, *username=''*, *password=''*, *pool_size=256*, *future_class=None*, *ssl_context=None*, *connector=None*, *loop=None*)
This function is responsible for instantiating the global variables that provide goblin connection configuration params.

Parameters

- **url** (*str*) – url for the Gremlin Server. Expected format: (ws|wss)://username:password@hostname:port/
- **pool_class** (*gremlinclient.pool.Pool*) – Pool class used to create global pool. If None trys to import tornado_client.Pool, if this import fails, trys to import aiohttp_client.Pool
- **graph_name** (*str*) – graph name as defined in server configuration. Defaults to "graph"
- **traversal_source** (*str*) – traversal source name as defined in the server configuration. Defaults to "g"
- **username** (*str*) – username as defined in the Tinkerpop credentials graph.
- **password** (*str*) – password for username as definined in the Tinkerpop credentials graph
- **pool_size** (*int*) – maximum number of connections allowed by global connection pool_size
- **future** (*class*) – type of Future. typically - asyncio.Future, trollius.Future, or tornado.concurrent.Future
- **ssl_context** (*ssl.SSLContext*) – ssl.SSLContext for secure protocol
- **connector** – connector used to establish gremlinclient connection. Overides ssl_context param.
- **loop** – io loop.

goblin.connection.**sync_spec**()

goblin.connection.**tear_down**()
Close the global connection pool.

goblin.constants module

goblin.exceptions module

exception goblin.exceptions.**ElementDefinitionException**
Bases: *goblin.exceptions.GoblinException*

Error in element definition

exception goblin.exceptions.**GoblinBlueprintsWrapperException**
Bases: *goblin.exceptions.GoblinException*

Exception thrown when a Blueprints wrapper error occurs

exception goblin.exceptions.**GoblinConnectionError**
Bases: *goblin.exceptions.GoblinException*

Problem connecting with Titan

exception goblin.exceptions.**GoblinException**
Bases: Exception

Generic Base Exception for Goblin Library

exception goblin.exceptions.**GoblinGraphMissingError**
Bases: *goblin.exceptions.GoblinException*

Graph with specified name does not exist

exception goblin.exceptions.**GoblinGremlinException**
Bases: *goblin.exceptions.GoblinException*

Exception thrown when a Gremlin error occurs

exception goblin.exceptions.**GoblinMetricsException**
Bases: *goblin.exceptions.GoblinException*

Exception thrown when a metric system error occurs

exception goblin.exceptions.**GoblinQueryError**
Bases: *goblin.exceptions.GoblinException*

Exception thrown when a query error occurs

exception goblin.exceptions.**GoblinRelationshipException**
Bases: *goblin.exceptions.GoblinException*

Exception thrown when a Relationship error occurs

exception goblin.exceptions.**ModelException**
Bases: *goblin.exceptions.GoblinException*

Error in model

exception goblin.exceptions.**SaveStrategyException**
Bases: *goblin.exceptions.GoblinException*

Exception thrown when a Save Strategy error occurs

exception goblin.exceptions.**ValidationError**(**args*, ***kwargs*)
Bases: *goblin.exceptions.GoblinException*

Exception thrown when a property value validation error occurs

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

g

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V