gclouddatastore Documentation

Release 0.1

JJ Geewax

January 22, 2014

Contents

1 Getting started **3**

2 API Documentation **5**
2.1 Getting started with Cloud Datastore 5
2.2 API Documentation 6

3 Indices and tables **17**

Python Module Index **19**

Warning: This library is still under construction and is **not** the official Google Python API client library. See the official documentation.

CHAPTER 1

Installing the library

The `gclouddatastore` library is `pip` install-able:

```
$ pip install gclouddatastore
```

CHAPTER 2

Getting started

A good place to look is the guide: *Getting started with Cloud Datastore*. If you have questions, or find any bugs, feel free to open an issue on the GitHub repository.

CHAPTER 3

API Documentation

Check out the full *API Documentation*.

3.1 Getting started with Cloud Datastore

3.1.1 Creating a project

Note: If you don't have a Google account, you should probably sign up for one now...

- **Create a project**

  Start off by visiting https://cloud.google.com/console and click on the big red button that says "Create Project".

- **Choose a name**

  In the box that says "name", choose something friendly. This is going to be the *human-readable* name for your project.

- **Choose an ID**

  In the box that says "ID", choose something unique (hyphens are OK). I typically choose a project name that starts with my initials, then a hyphen, then a unique identifier for the work I'm doing. For this example, you might choose `<initials>-quickstart`.

Then click OK (give it a second to create your project).

3.1.2 Enable the Cloud Datastore API

Now that you created a project, you need to *turn on* the Cloud Datastore API. This is sort of like telling Google which services you intend to use for this project.

- **Click on APIs & Auth** on the left hand side, and scroll down to where it says "Google Cloud Datastore API".
- **Click the "Off" button** on the right side to turn it into an "On" button.

3.1.3 Enable a "Service Account"

Now that you have a project that has access to the Cloud Datastore API, we need to make sure we are able to access our data. There are many ways to authenticate, but we're going to use a Service Account for today.

A *Service Account* is sort of like a username and password (like when you're connecting to your MySQL database), except the username is automatically generated (and is an e-mail address) and the password is actually a private key file.

To create a Service Account:

- **Click on Credentials** under the "APIs & Auth" section.
- **Click the big red button** that says "Create New Client ID" under the OAuth section (the first one).
- **Choose "Service Account"** and click the blue button that says "Create Client ID".
- **This will automatically** download a private key file. **Do not lost this.**
- **Rename your key** something shorter. I like to name the key `<project name>.key`.

  This is like your password for the account.
- **Copy the long weird e-mail address** labeled "E-mail address" in the information section for the Service Account you just created.

  This is like your username for the account.

OK. That's it! Time to start doing things with your Cloud Datastore project.

3.1.4 Add some data to your Datastore

Open a Python console and...

```
>>> import gclouddatastore
>>> dataset = gclouddatastore.get_dataset('<your-project-id-here',
        '<the e-mail address you copied here>',
        '/path/to/<your project>.key')
>>> dataset.query().fetch()
[]
>>> entity = dataset.entity('Person')
>>> entity['name'] = 'Your name'
>>> entity['age'] = 25
>>> entity.save()
>>> dataset.query('Person').fetch()
[<Entity{...} {'name': 'Your name', 'age': 25}>]
```

3.1.5 OK, go build cool stuff now

:)

3.2 API Documentation

3.2.1 gclouddatastore

Shortcut methods for getting set up with Google Cloud Datastore.

You'll typically use these to get started with the API:

```
>>> import gclouddatastore
>>> dataset = gclouddatastore.get_dataset('dataset-id-here',
                                          'long-email@googleapis.com',
                                          '/path/to/private.key')
>>> # Then do other things...
>>> query = dataset.query().kind('EntityKind')
>>> entity = dataset.entity('EntityKind')
```

The main concepts with this API are:

- gclouddatastore.connection.Connection which represents a connection between your machine and the Cloud Datastore API.
- gclouddatastore.dataset.Dataset which represents a particular dataset (akin to a database name in relational database world).
- gclouddatastore.entity.Entity which represents a single entity in the datastore (akin to a row in relational database world).
- gclouddatastore.key.Key which represents a pointer to a particular entity in the datastore (akin to a unique identifier in relational database world).
- gclouddatastore.query.Query which represents a lookup or search over the rows in the datastore.

gclouddatastore.__init__.**get_connection**(*client_email*, *private_key_path*)

Shortcut method to establish a connection to the Cloud Datastore.

Use this if you are going to access several datasets with the same set of credentials (unlikely):

```
>>> import gclouddatastore
>>> connection = gclouddatastore.get_connection(email, key_path)
>>> dataset1 = connection.dataset('dataset1')
>>> dataset2 = connection.dataset('dataset2')
```

Parameters

- **client_email** (*string*) – The e-mail attached to the service account.
- **private_key_path** (*string*) – The path to a private key file (this file was given to you when you created the service account).

Return type gclouddatastore.connection.Connection

Returns A connection defined with the proper credentials.

gclouddatastore.__init__.**get_dataset**(*dataset_id*, *client_email*, *private_key_path*)

Shortcut method to establish a connection to a particular dataset in the Cloud Datastore.

You'll generally use this as the first call to working with the API:

```
>>> import gclouddatastore
>>> dataset = gclouddatastore.get_dataset('dataset-id', email, key_path)
>>> # Now you can do things with the dataset.
>>> dataset.query().kind('TestKind').fetch()
[...]
```

Parameters

- **dataset_id** (*string*) – The id of the dataset you want to use. This is akin to a database name and is usually the same as your Cloud Datastore project name.
- **client_email** (*string*) – The e-mail attached to the service account.

- **private_key_path** (*string*) – The path to a private key file (this file was given to you when you created the service account).

Return type gclouddatastore.dataset.Dataset

Returns A dataset with a connection using the provided credentials.

3.2.2 Connections

class gclouddatastore.connection.**Connection**(*credentials=None*)
Bases: object

A connection to the Google Cloud Datastore via the Protobuf API.

This class should understand only the basic types (and protobufs) in method arguments, however should be capable of returning advanced types.

Parameters credentials (gclouddatastore.credentials.Credentials) – The OAuth2 Credentials to use for this connection.

API_BASE_URL = 'https://www.googleapis.com'
The base of the API call URL.

API_URL_TEMPLATE = '{api_base}/datastore/{api_version}/datasets/{dataset_id}/{method}'
A template used to craft the URL pointing toward a particular API call.

API_VERSION = 'v1beta2'
The version of the API, used in building the API call's URL.

classmethod build_api_url(*dataset_id*, *method*, *base_url=None*, *api_version=None*)
Construct the URL for a particular API call.

This method is used internally to come up with the URL to use when making RPCs to the Cloud Datastore API.

Parameters

- **dataset_id** (*string*) – The ID of the dataset to connect to. This is usually your project name in the cloud console.
- **method** (*string*) – The API method to call (ie, runQuery, lookup, ...).
- **base_url** (*string*) – The base URL where the API lives. You shouldn't have to provide this.
- **api_version** (*string*) – The version of the API to connect to. You shouldn't have to provide this.

dataset(**args*, ***kwargs*)
Factory method for Dataset objects.

Parameters args – All args and kwargs will be passed along to the gclouddatastore.dataset.Dataset initializer.

Return type gclouddatastore.dataset.Dataset

Returns A dataset object that will use this connection as its transport.

delete_entities(*dataset_id*, *key_pbs*)
Delete keys from a dataset in the Cloud Datastore.

This method deals only with gclouddatastore.datastore_v1_pb2.Key protobufs and not with any of the other abstractions. For example, it's used under the hood in the gclouddatastore.entity.Entity.delete() method.

Parameters

- **dataset_id** (*string*) – The dataset from which to delete the keys.
- **key_pbs** (list of `gclouddatastore.datastore_v1_pb2.Key` (or a single Key)) – The key (or keys) to delete from the datastore.

delete_entity(*dataset_id*, *key_pb*)

http

A getter for the HTTP transport used in talking to the API.

Return type `httplib2.Http`

Returns A Http object used to transport data.

lookup(*dataset_id*, *key_pbs*)

Lookup keys from a dataset in the Cloud Datastore.

This method deals only with protobufs (`gclouddatastore.datastore_v1_pb2.Key` and `gclouddatastore.datastore_v1_pb2.Entity`) and is used under the hood for methods like `gclouddatastore.dataset.Dataset.get_entity()`:

```
>>> import gclouddatastore
>>> from gclouddatastore.key import Key
>>> connection = gclouddatastore.get_connection(email, key_path)
>>> dataset = connection.dataset('dataset-id')
>>> key = Key(dataset=dataset).kind('MyKind').id(1234)
```

Using the `gclouddatastore.dataset.Dataset` helper:

```
>>> dataset.get_entity(key)
<Entity object>
```

Using the `connection` class directly:

```
>>> connection.lookup('dataset-id', key.to_protobuf())
<Entity protobuf>
```

Parameters

- **dataset_id** (*string*) – The dataset to look up the keys.
- **key_pbs** (list of `gclouddatastore.datastore_v1_pb2.Key` (or a single Key)) – The key (or keys) to retrieve from the datastore.

Return type list of `gclouddatastore.datastore_v1_pb2.Entity` (or a single Entity)

Returns The entities corresponding to the keys provided. If a single key was provided and no results matched, this will return None. If multiple keys were provided and no results matched, this will return an empty list.

run_query(*dataset_id*, *query_pb*, *namespace=None*)

Run a query on the Cloud Datastore.

Given a Query protobuf, sends a `runQuery` request to the Cloud Datastore API and returns a list of entity protobufs matching the query.

You typically wouldn't use this method directly, in favor of the `gclouddatastore.query.Query.fetch()` method.

Under the hood, the `gclouddatastore.query.Query` class uses this method to fetch data:

```
>>> import gclouddatastore
>>> connection = gclouddatastore.get_connection(email, key_path)
>>> dataset = connection.dataset('dataset-id')
>>> query = dataset.query().kind('MyKind').filter('property =', 'value')
```

Using the *fetch*‘ method...

```
>>> query.fetch()
[<list of Entity objects>]
```

Under the hood this is doing...

```
>>> connection.run_query('dataset-id', query.to_protobuf())
[<list of Entity Protobufs>]
```

Parameters

- **dataset_id** (*string*) – The ID of the dataset over which to run the query.
- **query_pb** (`gclouddatastore.datastore_v1_pb2.Query`) – The Protobuf representing the query to run.
- **namespace** (*string*) – The namespace over which to run the query.

save_entity(*dataset_id*, *key_pb*, *properties*)

3.2.3 Credentials

A simple wrapper around the OAuth2 credentials library.

class `gclouddatastore.credentials.`**Credentials**
Bases: `object`

An object used to simplify the OAuth2 credentials library.

Note: You should not need to use this class directly. Instead, use the helper methods provided in `gclouddatastore.__init__.get_connection()` and `gclouddatastore.__init__.get_dataset()` which use this class under the hood.

SCOPE = ‘https://www.googleapis.com/auth/datastore https://www.googleapis.com/auth/userinfo.email’
The scope required for authenticating as a Cloud Datastore consumer.

classmethod get_for_service_account(*client_email*, *private_key_path*)
Gets the credentials for a service account.

Parameters

- **client_email** (*string*) – The e-mail attached to the service account.
- **private_key_path** (*string*) – The path to a private key file (this file was given to you when you created the service account).

3.2.4 Datasets

class `gclouddatastore.dataset.`**Dataset**(*id*, *connection=None*)
Bases: `object`

connection()

entity(*kind*)

get_entities(*keys*)

get_entity(*key*)
Retrieves an entity from the dataset, along with all of its attributes.

Parameters **item_name** – The name of the item to retrieve.

Return type gclouddatastore.entity.Entity or None

Returns The requested entity, or None if there was no match found.

id()

query(**args*, ***kwargs*)

3.2.5 Entities

Class for representing a single entity in the Cloud Datastore.

Entities are akin to rows in a relational database, storing the actual instance of data.

Each entity is officially represented with a gclouddatastore.key.Key class, however it is possible that you might create an Entity with only a partial Key (that is, a Key with a Kind, and possibly a parent, but without an ID).

Entities in this API act like dictionaries with extras built in that allow you to delete or persist the data stored on the entity.

class gclouddatastore.entity.**Entity**(*dataset=None*, *kind=None*)
Bases: dict

Parameters

- **dataset** (gclouddatastore.dataset.Dataset) – The dataset in which this entity belongs.
- **kind** (*string*) – The kind of entity this is, akin to a table name in a relational database.

Entities are mutable and act like a subclass of a dictionary. This means you could take an existing entity and change the key to duplicate the object.

This can be used on its own, however it is likely easier to use the shortcut methods provided by gclouddatastore.dataset.Dataset such as:

- gclouddatastore.dataset.Dataset.entity() to create a new entity.

```
>>> dataset.entity('MyEntityKind')
<Entity[{'kind': 'MyEntityKind'}] {}>
```

- gclouddatastore.dataset.Dataset.get_entity() to retrive an existing entity.

```
>>> dataset.get_entity(key)
<Entity[{'kind': 'EntityKind', id: 1234}] {'property': 'value'}>
```

You can the set values on the entity just like you would on any other dictionary.

```
>>> entity['age'] = 20
>>> entity['name'] = 'JJ'
>>> entity
<Entity[{'kind': 'EntityKind', id: 1234}] {'age': 20, 'name': 'JJ'}>
```

And you can cast an entity to a regular Python dictionary with the *dict* builtin:

```
>>> dict(entity)
{'age': 20, 'name': 'JJ'}
```

dataset()
Get the gclouddatastore.dataset.Dataset in which this entity belonds.

Note: This is based on the gclouddatastore.key.Key set on the entity. That means that if you have no key set, the dataset might be *None*. It also means that if you change the key on the entity, this will refer to that key's dataset.

---

delete()
Delete the entity in the Cloud Datastore.

Note: This is based entirely off of the gclouddatastore.key.Key set on the entity. Whatever is stored remotely using the key on the entity will be deleted.

---

classmethod from_key(*key*)
Factory method for creating an entity based on the gclouddatastore.key.Key.

Parameters **key** (gclouddatastore.key.Key) – The key for the entity.

Returns The Entity derived from the gclouddatastore.key.Key.

classmethod from_protobuf(*pb*, *dataset=None*)
Factory method for creating an entity based on a protobuf.

The protobuf should be one returned from the Cloud Datastore Protobuf API.

Parameters **key** (gclouddatastore.datastore_v1_pb2.Entity) – The Protobuf representing the entity.

Returns The Entity derived from the gclouddatastore.datastore_v1_pb2.Entity.

key(*key=None*)
Get or set the gclouddatastore.key.Key on the current entity.

Parameters **key** (glcouddatastore.key.Key) – The key you want to set on the entity.

Returns Either the current key or the Entity.

```
>>> entity.key(my_other_key)  # This returns the original entity.
<Entity[{'kind': 'OtherKeyKind', 'id': 1234}] {'property': 'value'}>
>>> entity.key()  # This returns the key.
<Key[{'kind': 'OtherKeyKind', 'id': 1234}]>
```

kind()
Get the kind of the current entity.

Note: This relies entirely on the gclouddatastore.key.Key set on the entity. That means that we're not storing the kind of the entity at all, just the properties and a pointer to a Key which knows its Kind.

---

reload()
Reloads the contents of this entity from the datastore.

This method takes the gclouddatastore.key.Key, loads all properties from the Cloud Datastore, and sets the updated properties on the current object.

> **Warning:** This will override any existing properties if a different value exists remotely, however it will *not* override any properties that exist only locally.

save()
Save the entity in the Cloud Datastore.

Return type gclouddatastore.entity.Entity

Returns The entity with a possibly updated Key.

3.2.6 Keys

class gclouddatastore.key.**Key**(*dataset=None*, *namespace=None*, *path=None*)
Bases: object

An immutable representation of a datastore Key.

dataset(*dataset=None*)

classmethod from_path(**args*, ***kwargs*)

classmethod from_protobuf(*pb*, *dataset=None*)

id(*id=None*)

id_or_name()

is_partial()

kind(*kind=None*)

name(*name=None*)

namespace(*namespace=None*)

parent()

path(*path=None*)

to_protobuf()

3.2.7 Queries

class gclouddatastore.query.**Query**(*kinds=None*, *dataset=None*)
Bases: object

A Query against the Cloud Datastore.

This class serves as an abstraction for creating a query over data stored in the Cloud Datastore.

Each Query object is immutable, and a clone is returned whenever any part of the query is modified:

```
>>> query = Query('MyKind')
>>> limited_query = query.limit(10)
>>> query.limit() == 10
False
>>> limited_query.limit() == 10
True
```

You typically won't construct a `Query` by initializing it like `Query('MyKind', dataset=...)` but instead use the helper `gclouddatastore.dataset.Dataset.query()` method which generates a query that can be executed without any additional work:

```
>>> import gclouddatastore
>>> dataset = gclouddatastore.get_dataset('dataset-id', email, key_path)
>>> query = dataset.query('MyKind')
```

Parameters

- **kinds** (*string or list of strings*) – The kind or kinds to query.
- **dataset** (`gclouddatastore.dataset.Dataset`) – The dataset to query.

OPERATORS = {'=': 5, '<=': 2, '>=': 4, '<': 1, '>': 3}
Mapping of operator strings and their protobuf equivalents.

dataset (*dataset=None*)
Get or set the `gclouddatastore.dataset.Dataset` for this Query.

This is the dataset against which the Query will be run.

This is a hybrid getter / setter, used as:

```
>>> query = Query('Person')
>>> query = query.dataset(my_dataset)  # Set the dataset.
>>> query.dataset()  # Get the current dataset.
<Dataset object>
```

Return type `gclouddatastore.dataset.Dataset`, None, or `Query`

Returns If no arguments, returns the current dataset. If a dataset is provided, returns a clone of the `Query` with that dataset set.

fetch (*limit=None*)
Executes the Query and returns all matching entities.

This makes an API call to the Cloud Datastore, sends the Query as a protobuf, parses the responses to Entity protobufs, and then converts them to `gclouddatastore.entity.Entity` objects.

For example:

```
>>> import gclouddatastore
>>> dataset = gclouddatastore.get_dataset('dataset-id', email, key_path)
>>> query = dataset.query('Person').filter('name =', 'Sally')
>>> query.fetch()
[<Entity object>, <Entity object>, ...]
>>> query.fetch(1)
[<Entity object>]
>>> query.limit()
None
```

Parameters **limit** (*integer*) – An optional limit to apply temporarily to this query. That is, the Query itself won't be altered, but the limit will be applied to the query before it is executed.

Return type list of `gclouddatastore.entity.Entity`'s

Returns The list of entities matching this query's criteria.

filter(*expression*, *value*)
Filter the query based on an expression and a value.

This will return a clone of the current `Query` filtered by the expression and value provided.

Expressions take the form of:

```
.filter('<property> <operator>', <value>)
```

where property is a property stored on the entity in the datastore and operator is one of `OPERATORS` (ie, =, <, <=, >, >=):

```
>>> query = Query('Person')
>>> filtered_query = query.filter('name =', 'James')
>>> filtered_query = query.filter('age >', 50)
```

Because each call to `.filter()` returns a cloned `Query` object we are able to string these together:

```
>>> query = Query('Person').filter('name =', 'James').filter('age >', 50)
```

Parameters

- **expression** (*string*) – An expression of a property and an operator (ie, =).
- **value** (*integer, string, boolean, float, None, datetime*) – The value to filter on.

Return type `Query`

Returns A Query filtered by the expression and value provided.

kind(**kinds*)
Get or set the Kind of the Query.

Note: This is an **additive** operation. That is, if the Query is set for kinds A and B, and you call `.kind('C')`, it will query for kinds A, B, *and*, C.

Parameters **kinds** (*string*) – The entity kinds for which to query.

Return type string or `Query`

Returns If no arguments, returns the kind. If a kind is provided, returns a clone of the `Query` with those kinds set.

limit(*limit=None*)
Get or set the limit of the Query.

This is the maximum number of rows (Entities) to return for this Query.

This is a hybrid getter / setter, used as:

```
>>> query = Query('Person')
>>> query = query.limit(100)  # Set the limit to 100 rows.
>>> query.limit()  # Get the limit for this query.
100
```

Return type integer, None, or `Query`

Returns If no arguments, returns the current limit. If a limit is provided, returns a clone of the `Query` with that limit set.

to_protobuf()
Convert the Query instance to a gclouddatastore.datastore_v1_pb2.Query.

Return type gclouddatstore.datastore_v1_pb2.Query

Returns A Query protobuf that can be sent to the protobuf API.

3.2.8 Helpers

Helper methods for dealing with Cloud Datastore's Protobuf API.

gclouddatastore.helpers.**get_protobuf_attribute_and_value**(*val*)
Given a value, return the protobuf attribute name and proper value.

The Protobuf API uses different attribute names based on value types rather than inferring the type. This method simply determines the proper attribute name based on the type of the value provided and returns the attribute name as well as a properly formatted value.

Certain value types need to be coerced into a different type (such as a *datetime.datetime* into an integer timestamp, or a *gclouddatastore.key.Key* into a Protobuf representation. This method handles that for you.

For example:

```
>>> get_protobuf_attribute_and_value(1234)
('integer_value', 1234)
>>> get_protobuf_attribute_and_value('my_string')
('string_value', 'my_string')
```

Parameters **val** (*datetime.datetime*, gclouddatastore.key.Key, bool, float, integer, string) – The value to be scrutinized.

Returns A tuple of the attribute name and proper value type.

gclouddatastore.helpers.**get_value_from_protobuf**(*pb*)
Given a protobuf for a Property, get the correct value.

The Cloud Datastore Protobuf API returns a Property Protobuf which has one value set and the rest blank. This method retrieves the the one value provided.

Some work is done to coerce the return value into a more useful type (particularly in the case of a timestamp value, or a key value).

Parameters **pb** (gclouddatastore.datastore_v1_pb2.Property) – The Property Protobuf.

Returns The value provided by the Protobuf.

CHAPTER 4

Indices and tables

- *genindex*
- *modindex*

g

gclouddatastore.__init__, 6
gclouddatastore.connection, 8
gclouddatastore.credentials, 10
gclouddatastore.dataset, 10
gclouddatastore.entity, 11
gclouddatastore.helpers, 16
gclouddatastore.key, 13
gclouddatastore.query, 13