Google App Engine
Python SDK 1.2.2

REVISION 1.1 05.21.09

Datastore

`google.appengine.ext.db`

A scalable storage and query engine.

PACKAGE FUNCTIONS

`get(key)`	**Model instance**

You can also pass multiple keys and it will return multiple Model instances.

`put(model_instance)`	**Key object**

You can also pass multiple model instances and it will return multiple keys.

```
delete(model_instance|key)
run_in_transaction(function, *args, **kwargs)
run_in_transaction_custom_retries(retries, function,
                                  *args, **kwargs)
```

Model is the superclass for data model definitions.

CONSTRUCTOR

```
class Model(parent=None, key_name=None, **kwds)
```

CLASS METHODS

`get(key)`	**Key object**
`get_by_id(id, parent=None)`	**Model instance**

You can also pass multiple ids and it will return multiple Model instances.

`get_by_key_name(key_name, parent=None)`	**see above**

You can also pass multiple ids and it will return multiple Model instances.

`get_or_insert(key_name, **kwds)`	**see above**
`all()`	**Query object**
`gql(query_string, *args, **kwds)`	**GQLQuery object**

Examples:

```
s = Story.gql("WHERE title = :1", "The Little Prince")
s = Story.gql("WHERE title = :title", title="The Little Prince")
```

`kind()`	**Kind**
`properties()`	**dict**

INSTANCE METHODS

`key()`	**Key**
`put()`	**Key**
`delete()`	
`is_saved()`	**bool**
`parent()`	**Model**
`parent_key()`	**Key**
`to_xml()`	**XML**

Property is the superclass for data model definitions.

CONSTRUCTOR

```
class Property(verbose_name=None, name=None,
          indexed=True, default=None, choices=None
          required=False, validator=None)
```

CLASS ATTRIBUTES

```
data_type
```

INSTANCE METHODS

`default_value()`	**value**
`validate(value)`	**value or exception**
`empty(value)`	**bool**

TYPE AND PROPERTY CLASSES

Property Class	Value Type	Sort Order
StringProperty	str unicode	Unicode (str is treated as ASCII)
ByteStringProperty	db.ByteString	byte order
BooleanProperty	bool	False < True
IntegerProperty	int long	Numeric
FloatProperty	float	Numeric
DateTimeProperty DateProperty TimeProperty	datetime.datetime	Chronological
ListProperty StringListProperty	list	If ascending, by least element; if descending, by greatest element
ReferenceProperty SelfReferenceProperty	db.Key	By path elements (kind, ID or name)
UserProperty	user.User	By email address
BlobProperty	db.Blob	(not orderable)
TextProperty	db.Text	(not orderable)
CategoryProperty	db.Category	Unicode
LinkProperty	db.Link	Unicode
EmailProperty	db.Email	Unicode
GeoPtProperty	db.GeoPt	By latitude, then longitude
IMProperty	db.IM	Unicode
PhoneNumberProperty	db.PhoneNumber	Unicode
PostalAddressProperty	db.PostalAddress	Unicode
RatingProperty	db.Rating	Unicode

Query uses objects and methods to prepare queries.

CONSTRUCTOR

```
class Query(model_class)
```

INSTANCE METHODS

`filter(property_operator, value)`	**self**
`order(property)`	**self**
`ancestor(model_instance|key)`	**self**
`get()`	**model instance or None**
`fetch(limit, offset=0)`	**list of model instances**
`count(limit=None)`	**integer**

Key represents a unique key for a datastore entity.

CONSTRUCTOR

```
class Key(encoded=None)
```

CLASS METHODS

```
Key.from_path(*args, **kwds)
```

This example creates a key for an Address entity with the numeric ID 9876 whose parent is a User entity with the named key 'Boris':

```
k = Key.from_path('User', 'Boris', 'Address', 9876)
```

INSTANCE METHODS

`app()`	**Application name (string)**
`kind()`	**Kind (string)**
`id()`	**Numeric ID (int)**
`name()`	**Entity name (string)**
`id_or_name()`	**Numeric ID(int) or name (string)**
`has_id_or_name()`	**bool**
`parent()`	**Key**

GQL is a SQL-like language for retrieving entities.

SYNTAX

```
WHERE <condition> [AND <condition> ...]
ORDER BY <property> [ASC | DESC] [,<property> [ASC | DESC]…]
LIMIT [<offset>,]<count>
OFFSET <offset>
  <condition> := <property> {< | <= | > | >= | = | != } <value>
  <condition> := <property> IN <list>
  <condition> := ANCESTOR IS <entity or key>
  <list> := (<value>, …)
```

Note that `:NUMBER` and `:NAME` are substitutions for positional and keyword arguments, referrring to `*args` (starting at 1) and `**kwds` respectively. See `Model.gql()` for example usage. Key-only queries are supported using either `SELECT __key__`.

Google

Memcache

`google.appengine.api.memcache`

A distributed in-memory data cache that can be used in front of or in place of persistent storage.

FUNCTIONS

`set(key, value, time=0, min_compress_len=0)`	**bool**

`True` means done while `False` means an error occured.

Note that a Memcache `key` is an arbitrary string, not an instance of `db.Key`.

`set_multi(mapping, time=0, key_prefix='', min_compress_len=0)`	**list**
`get(key)`	**value**
`get_multi(keys, key_prefix='')`	**dict**
`delete(key, seconds=0)`	**error code**
`delete_multi(keys, seconds=0, key_prefix='')`	**bool**
`add(key, value, time=0, min_compress_len=0)`	**bool**
`add_multi(mapping, time=0, key_prefix='', min_compress_len=0)`	**list**
`replace(key, value, time=0, min_compress_len=0)`	**bool**
`replace_multi(mapping, time=0, key_prefix='', min_compress_len=0)`	**list**
`incr(key, delta=1)`	**int, long or None**
`decr(key, delta=1)`	**int, long or None**
`flush_all()`	**bool**
`get_stats()`	**dict**

User

`google.appengine.api.users`

An App Engine application can redirect a user to a Google Accounts page to sign in register, or sign out.

User represents a user with a Google account.

CONSTRUCTOR

`class User(email=None)`

INSTANCE METHODS

`email()`	**string**
`nickname()`	**string**
`user_id()`	**string**

This can be the user id of an email address or the full email address if it differs from the application's auth domain (gmail.com or the Google Apps domain for which the application is registered).

FUNCTIONS

`create_login_url(dest_url)`	**string (URL)**
`create_logout_url(dest_url)`	**string (URL)**
`get_current_user()`	**User**
`is_current_user_admin()`	**bool**

EXCEPTIONS

`Error, UserNotFound(), RedirectTooLongError()`

URL Fetch

`google.appengine.api.urlfetch`

The URLFetch API can retrieve data using HTTP and HTTPS URLs.

FUNCTIONS

`fetch(url, payload=None, method=GET, headers=(), allow_truncated=False, follow_redirects=True, deadline=5)`	**Response object**

RESPONSE OBJECTS

`content`

The body content of the response.

`content_was_truncated`

True if the `allow_truncated` parameter to `fetch()` was `True` and the response exceeded the maximum response size. In this case, the content attribute contains the truncated response.

`status_code`

The HTTP status code.

`headers`

The HTTP response headers, as a mapping of names to values.

EXCEPTION CLASSES

`Error, InvalidURLError, DownloadError, ResponseTooLargeError`

Mail

`google.appengine.api.urlfetch`

Provides two ways to send an email message: the mail.send_mail() function and the EmailMessage class.

EmailMessage represents an email message.

CONSTRUCTOR

`class EmailMessage(**fields)`

INSTANCE METHODS

`check_initialized()`	
`initialize(**fields)`	
`is_initialized()`	**bool**
`send()`	

FUNCTIONS

`check_email_valid(email_address, field)`

This raises an `InvalidEmailError` when the `email_address` is invalid.

`invalid_email_reason(email_address, field)`	**string**
`is_email_valid(email_address)`	**Boolean**
`send_mail(sender, to, subject, body, **kw)`	
`send_mail_to_admins(sender, subject, body, **kw)`	

EXCEPTIONS

`Error, BadRequestError, InvalidEmailError, InvalidAttachmentTypeError, MissingRecipientsError, MisssingSenderError, MissingSubjectError, MissingBodyError`

MESSAGE FIELDS (**fields)

`sender, to, cc, bcc, reply_to, subject, body, html, attachments`

Images

`google.appengine.api.images`

Provides image manipulation using the Picassa Web infrastructure.

Image represents image data to be transformed.

CONSTRUCTOR

`class Image(image_data)`

PROPERTIES

`width, height`

INSTANCE METHODS

`resize(width=0, height=0)`

`crop(left_x, top_y, right_x, bottom_y)`

The four number arguments are multiplied by the image's width and height to define a bounding box that crops the image. The upper left point of the bounding box is at (left_x*image_width, top_y*image_height) the lower right point is at (right_x*image_width, bottom_y*image_height).

`rotate(clockwise_degrees)`	
`horizontal_flip()`	
`vertical_flip()`	
`im_feeling_lucky()`	
`composite(inputs, width, height, color=0, output_encoding=images.PNG)`	
`histogram(image_data)`	**list**
`execute_transforms()`	**Image Object**

FUNCTIONS

They are the same as the instance methods, but they can be performed directly on image_data. There is no need to queue them using execute_transforms(). They include an additional parameter which is the expected `output_encoding` image type, which defaults to PNG.

EXCEPTIONS

`Error, TransformationError, BadRequestError, NotImageError, BadImageError, LargeImageError`