



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,
               default=None, required=False,
               validator=None, choices=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	datetime.datetime	Chronological
DateProperty		
TimeProperty		
ListProperty	list	If ascending, by least element; if descending, by greatest element
StringListProperty		
ReferenceProperty	db.Key	By path elements (kind, ID or name)
SelfReferenceProperty		
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. Model.all() returns a query object.

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)
```

The following example creates a key for an entity of kind Address with the numeric ID 9876 whose parent is an entity of kind User 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, referring to *args (starting at 1) and **kwds respectively. See Model.gql() for example usage.

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

<code>set(key, value, time=0, min_compress_len=0)</code>	bool
True means done while False means an error occurred.	
Note that a Memcache key is an arbitrary string, not an instance of <code>db.Key</code> .	
<code>set_multi(mapping, time=0, key_prefix='', min_compress_len=0)</code>	list
<code>get(key)</code>	value
<code>get_multi(keys, key_prefix='')</code>	dict
<code>delete(key, seconds=0)</code>	error code
<code>delete_multi(keys, seconds=0, key_prefix='')</code>	bool
<code>add(key, value, time=0, min_compress_len=0)</code>	bool
<code>add_multi(mapping, time=0, key_prefix='', min_compress_len=0)</code>	list
<code>replace(key, value, time=0, min_compress_len=0)</code>	bool
<code>replace_multi(mapping, time=0, key_prefix='', min_compress_len=0)</code>	list
<code>incr(key, delta=1)</code>	int, long or None
<code>decr(key, delta=1)</code>	int, long or None
<code>flush_all()</code>	bool
<code>get_stats()</code>	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

<code>email()</code>	string
<code>nickname()</code>	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

<code>create_login_url(dest_url)</code>	string (URL)
<code>create_logout_url(dest_url)</code>	string (URL)
<code>get_current_user()</code>	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, Response object,  
headers=(), allow_truncated=False,  
follow_redirects=True)
```

RESPONSE OBJECTS

<code>content</code>	The body content of the response.
<code>content_was_truncated</code>	True if the <code>allow_truncated</code> parameter to <code>fetch()</code> was True and the response exceeded the maximum response size. In this case, the <code>content</code> attribute contains the truncated response.

<code>status_code</code>	The HTTP status code.
<code>headers</code>	The HTTP response headers, as a mapping of names to values.

EXCEPTION CLASSES

```
Error, InvalidURLError, DownloadError,  
ResponseTooLargeError
```

Mail

[google.appengine.api.urlfetch](#)

The mail API 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

<code>check_initialized()</code>	
<code>initialize(**fields)</code>	
<code>is_initialized()</code>	bool
<code>send()</code>	

FUNCTIONS

<code>check_email_valid(email_address, field)</code>	This raises an <code>InvalidEmailError</code> when the <code>email_address</code> is invalid.
<code>invalid_email_reason(email_address, field)</code>	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,  
MissingSenderError, MissingSubjectError,  
MissingBodyError
```

MESSAGE FIELDS (**fields)

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

Images

[google.appengine.api.images](#)

The App Engine image service lets your application manipulate images using the same infrastructure as Picasa Web.

■ **Image** represents image data to be transformed.

CONSTRUCTOR

```
class Image(image_data)
```

PROPERTIES

```
width, height
```

INSTANCE METHODS

<code>resize(width=0, height=0)</code>	
<code>crop(left_x, top_y, right_x, bottom_y)</code>	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).

<code>rotate(clockwise_degrees)</code>	
<code>horizontal_flip()</code>	
<code>vertical_flip()</code>	
<code>im_feeling_lucky()</code>	
<code>execute_transforms(out_encoding=images.PNG JPEG)</code>	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 `out_encoding` image type, which defaults to PNG.

EXCEPTIONS

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