**Datastore**
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)  

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

- 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

- get_or_insert(key_name, **kwds)  
  see above

- all()  
  Query object

- gql(query_string, *args, **kwds)  
  GQLQuery object

Examples:

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

**CLASS ATTRIBUTES**

- data_type

**INSTANCE METHODS**

- default_value()
  value

- validate(value)
  value or exception

- empty(value)
  bool

**TYPE AND PROPERTY CLASSES**

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

**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':

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

**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> := IN <list>
- <condition> := ANCESTOR IS <entity or key>

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. Key-only queries are supported using either SELECT __key__.  

---

Google App Engine  
Python SDK 1.2.2  
REVISION 1.1 05.21.09
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 occurred.

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

set_multi(mapping, time=0, key_prefix='') list

get(key) value

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.

**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

Exception

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.

**CONSTRUCTOR**

class EmailMessage(**fields)

**INSTANCE METHODS**

check_initialized() boolean

initialize(**fields)

is_initialized() bool

send()

FUNCTIONS

check_email_valid(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, MissingSenderError, 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.

**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