NDB Cheat Sheet

rodrigo.moraes@gmail.com, guido@google.com
(last update: 8/7/2013)

Cheat Sheet: ext.db to ndb

The tables below show similarities and differences between ndb and the old ext.db module. See the Official NDB Docs for an introduction to and reference for NDB. You may also be interested in a blog entry by Khan Academy intern Dylan Vassallo about upgrading models to NDB.

No Datastore Changes Needed!

In case you wondered, despite the different APIs, NDB and the old ext.db package write *exactly* the same data to the Datastore. That means you don't have to do any conversion to your datastore, and you can happily mix and match NDB and ext.db code, as long as the schema you use is equivalent. You can even convert between ext.db and NDB keys using ndb.Key.from_old_key() and key.to_old_key().

General differences

- NDB is picky about types. E.g. in db, when a key is required, you can also pass an entity or a string. In NDB you must pass a key.
- NDB is picky about lists. E.g. in db, db.put() takes either an entity or a list of entities. In NDB, you use entity.put() to put a single entity, but ndb.put_multi(<list>) to put a list of entities.
- NDB prefers methods over functions. E.g. instead of db.get(key), and db.put(entity), NDB uses key.get() and entity.put().
- NDB doesn't like offering two APIs that do the same thing. (On the other hand it does sometimes offer two APIs that do slightly different things.)

[TBD: Maybe also list some things that are the same?]
[TBD: What else is part of the synchronous API?]
[TBD: Add direct links to docs in table]

Model class

google.appengine.ext.db	**ndb.model**
`class MyModel(db.Model):` `  foo = db.StringProperty()`	`class MyModel(ndb.Model):` `  foo = ndb.StringProperty()`

`@classmethod`
`def kind(cls):`
`  return 'Foo'`
	`@classmethod`
	`def _get_kind(cls):`
	`  return 'Foo'`

`MyModel.kind()`	`MyModel._get_kind()`
`MyModel.properties()` `model_instance.properties()`	`MyModel._properties  # No () !!` `model_entity._properties`
`MyExpando.dynamic_properties()`	`MyExpando._properties  # No () !!`

Entities

google.appengine.ext.db	**ndb.model**
`MyModel(key_name='my_key')`	`MyModel(id='my_key')`
`MyModel(key_name='my_key',` `  parent=model_instance)`	`MyModel(id='my_key',` `  parent=model_instance.key)`
`key = model_instance.key()`	`key = model_instance.key  # no () !!`

`model_instance = MyModel(`
`  foo='foo',`
`  bar='bar',`
`  baz='baz')`
	`model_instance = MyModel(`
	`  foo='foo',`
	`  bar='bar',`
	`  baz='baz')`

`model_instance.foo = 'foo'`
`model_instance.bar = 'bar'`
`model_instance.baz = 'baz'`
	`model_instance.foo = 'foo'`
	`model_instance.bar = 'bar'`
	`model_instance.baz = 'baz'`
	`# or a shortcut...`
	`model_instance.populate(`
	`  foo='foo',`
	`  bar='bar',`
	`  baz='baz')`

`model_instance.is_saved()`	`# No direct equivalent; see` http://stackoverflow.com/questions/12083254/is-it-possible-to-determine-with-ndb-if-model-is-persistent-in-the-datastore-or/12096066#12096066 `for a possible solution`

Get

google.appengine.ext.db	ndb.model
`MyModel.get_by_key_name('my_key')`	`MyModel.get_by_id('my_key')`
`MyModel.get_by_id(42)`	`MyModel.get_by_id(42)`
`db.get(key)`	`key.get()`
`MyModel.get(key)`	`key.get()`
`db.get(model_instance)`	`model_instance.key.get()`
`db.get(list_of_keys)`	`ndb.get_multi(list_of_keys)`
`db.get(list_of_instances)`	`ndb.get_multi([x.key for x in list_of_instances])`
`MyModel.get_or_insert('my_key', parent=model_instance, foo='bar')`	`MyModel.get_or_insert('my_key', parent=model_instance.key, foo='bar')`

Put

google.appengine.ext.db	ndb.model
`db.put(model_instance)`	`model_instance.put()`
`db.put(list_of_model_instances)`	`ndb.put_multi( list_of_model_instances)`

Delete

google.appengine.ext.db	ndb.model
`model_instance.delete()`	`model_instance.key.delete()`
`db.delete(model_instance)`	`model_instance.key.delete()`
`db.delete(key)`	`key.delete()`
`db.delete(list_of_model_instances)`	`ndb.delete_multi([m.key for m in list_of_model_instances])`

db.delete(list_of_keys)	ndb.delete_multi(list_of_keys)

Properties

google.appengine.ext.db	ndb.model
db.BlobProperty()	ndb.BlobProperty()
db.BooleanProperty()	ndb.BooleanProperty()
db.ByteStringProperty()	ndb.BlobProperty(indexed=True)
db.CategoryProperty()	ndb.StringProperty()
db.DateProperty()	ndb.DateProperty()
db.DateTimeProperty()	ndb.DateTimeProperty()
db.EmailProperty()	ndb.StringProperty()
db.FloatProperty()	ndb.FloatProperty()
db.GeoPtProperty()	ndb.GeoPtProperty()
db.IMProperty()	# No equivalent
db.IntegerProperty()	ndb.IntegerProperty()
db.LinkProperty()	ndb.StringProperty() (but beware the max size of 500 -- if you have longer urls, use ndb.TextProperty())
db.ListProperty(bool) db.ListProperty(float) db.ListProperty(int) db.ListProperty(db.Key) # etc.	ndb.BooleanProperty(repeated=True) ndb.FloatProperty(repeated=True) ndb.IntegerProperty(repeated=True) ndb.KeyProperty(repeated=True) # etc.
db.PhoneNumberProperty()	ndb.StringProperty()
db.PostalAddressProperty()	ndb.StringProperty()
db.RatingProperty()	ndb.IntegerProperty()
db.ReferenceProperty(AnotherModel) model_instance.prop	ndb.KeyProperty(kind=AnotherModel) model_instance.prop.get()

`MyModel.prop \`
`  .get_value_for_datastore \`
`  (model_instance)`
	`model_instance.prop`

`# Using the backreference set`
`other = model_instance.prop`
`other.prop_set.fetch(N)`
	`# No direct equivalent; emulation:`
	`other = model_instance.prop.get()`
	`MyModel.query(`
	`  MyModel.prop == other.key).fetch(N)`

`db.SelfReferenceProperty()`	`ndb.KeyProperty(kind='ThisModelClass')`
`db.StringProperty()`	`ndb.StringProperty()`
`db.StringProperty(multiline=True)`	`# Not supported; strings are always` `# allowed to contain '\n'`
`db.StringListProperty()`	`ndb.StringProperty(repeated=True)`
`db.TextProperty()`	`ndb.TextProperty()`
`db.TimeProperty()`	`ndb.TimeProperty()`
`db.UserProperty()`	`ndb.UserProperty()`
`blobstore.BlobReferenceProperty()`	`ndb.BlobKeyProperty()`

Building a Key

google.appengine.ext.db	**ndb.model**
`key = db.Key(encoded_key)`	`key = ndb.Key(urlsafe=encoded_key)`

`key = db.Key.from_path(`
`  'MyKind', 'some_id',`
`  'MyKind', 'some_id')`
	`key = ndb.Key(`
	`  'MyKind', 'some_id',`
	`  'MyKind', 'some_id')`

`key = db.Key.from_path(`
`  MyModel, 'some_id',`
`  parent=model_instance,`
`  namespace='my_namespace')`
	`key = ndb.Key(`
	`  MyModel, 'some_id',`
	`  parent=model_instance.key,`
	`  namespace='my_namespace')`

Key operations

google.appengine.ext.db	ndb.model
`key.id_or_name()`	`key.id()`
`key.id()`	`key.integer_id()`
`key.name()`	`key.string_id()`

`key.has_id_or_name()`	`key.id() is None`
	`# or...`
	`model_instance.has_complete_key()`

`key.app(), key.namespace(), key.parent(), key.kind()`	`# same thing`
`str(key)`	`key.urlsafe()`
`key.to_path()`	`key.flat()`
`db.allocate_ids(MyModel, size)`	`S, E = MyModel.allocate_ids(size)`
`db.allocate_id_range(MyModel,X,Y)`	`S, E = MyModel.allocate_ids(max=Y)` `assert S <= X`

Transactions

google.appengine.ext.db	ndb.model
`db.run_in_transaction(function)`	`ndb.transaction(function)`
`db.run_in_transaction(` `  function, *args, **kwds)`	`ndb.transaction(` `  lambda: function(*args, **kwds))`
`db.run_in_transaction_custom_retries(n, function)`	`ndb.transaction(function, retries=n)`

`opts = \`
`  db.create_transaction_options(xg=True)`
`db.run_in_transaction_options(opts, fun)`
	`ndb.transaction(fun, xg=True)`

Queries

google.appengine.ext.db	ndb.model

`q = MyModel.all()`	`q = MyModel.query()`
`for result in q.run(): ...`	`for result in q.iter(): ...`

`q = MyModel.all() \`
`  .filter('foo =', 'bar') \`
`  .filter('baz >=', 'ding')`
	`q = MyModel.query(`
	`  MyModel.foo == 'bar',`
	`  MyModel.baz >= 'ding')`

`q = MyModel.all()`
`q.filter('foo =', 'bar')`
`q.filter('baz >=', 'ding')`
`q.order('-foo')`
`results = q.fetch(10)`
	`q = MyModel.query()`
	`q = q.filter(MyModel.foo == 'bar')`
	`q = q.filter(MyModel.baz >= 'ding')`
	`q = q.order(-MyModel.foo)`
	`results = q.fetch(10)`

`q.filter('__key__', k)` `# k is a db.Key instance`	`q = q.filter(MyModel._key == k)` `# k is an ndb.Key instance`
`a.filter('__key__ >=', k)` `# k is a db.Key instance`	`q = q.filter(MyModel._key >= k)` `# k is an ndb.Key instance`

`class MyExpando(Expando): pass`
`q = MyExpando.all()`
`q.filter('foo =', 'bar')`
	`class MyExpando(Expando): pass`
	`q = MyExpando.query(`
	` ndb.GenericProperty('foo') == 'bar')`

`class Foo(Model): ...`
`class Bar(Model):`
`  foo = ReferenceProperty(Foo)`
`myfoo = <some Foo instance>`
`for bar in myfoo.bar_set(): ...`
	`class Foo(Model): ...`
	`class Bar(Model):`
	`  foo = KeyProperty(kind=Foo)`
	`myfoo = <some Foo instance>`
	`for bar in \`
	`  Bar.query(Bar.foo == myfoo.key): ...`

`q = MyModel.all()` `q.ancestor(ancestor_key)`	`q =` `  MyModel.query(ancestor=ancestor_key)`

`q = MyModel.all(keys_only=True)`
`r = q.fetch(N)`
	`r = MyModel.query() \`
	`  .fetch(N, keys_only=True)`
	`# Alternatively:`
	`q = MyModel.query(`
	`  default_options=QueryOptions(`
	`  keys_only=True))`
	`r = q.fetch(N)`

`q = MyModel.gql(...)`	`# same thing`

Cursors

`q = MyModel.all()` `a = q.fetch(20)`	`q = MyModel.query()` `a, cur, more = q.fetch_page(20)`

`cur = q.cursor()`	`# (1)`
`q.with_cursor(cur)` `b = q.fetch(20)`	`b, cur, more = \` `  q.fetch_page(20, start_cursor=cur)`
`q.with_cursor(end_cursor=cur)` `b = q.fetch(20)`	`q.fetch(20, end_cursor=cur)`

(1) In NDB, more is a bool indicating whether there are more entities at the cursor.