GridFS Support in Ming

One caveat to MongoDB's flexible storage model comes up when you start wanting to store large files in collections: MongoDB has a hard limit of 16MB per document. Fortunately, PyMongo (and Ming) have built-in support for gridfs, a convention that allows us to split up data into 'chunks' and access them as a single 'file.'

For a given gridfs 'filesystem', there is a root collection name (default `fs`) and two names derived from the root, the files and chunks collections. In the default case, then, the two collections would be `fs.files` and `fs.chunks`. The purpose of `fs.files` is to store metadata about the files in the filesystem. The purpose of `fs.chunks` is to store the actual data content of the files.

Ming provides support for treating the `fs.files` collection as a regular `ming.collection` using the `ming.fs` module and the `ming.fs.filesystem` constructor. In the simplest case, we want to use gridfs as a 'bit bucket' for storing large data objects with no optional metadata:

In [1]:
```
import ming
from ming import fs

from lesson_2_0 import model as M20

sess = M20.sess

Attachment = fs.filesystem(
    'forum.attachment', sess)

Attachment.m.fs
```

Out[1]:
```
<gridfs.GridFS at 0x2d89090>
```

Note that PyMongo's underlying `GridFS` instance is available as we would expect on the class manager. Now, let's create a file in the Attachment filesystem:

In [2]:
```
# Make sure we're dealing with a clean database
def clean_database():
    sess.db.forum.attachment.files.remove()
    sess.db.forum.attachment.chunks.remove()
clean_database()

# Insert a file and show the metadata using a regular Ming
#  query
a = Attachment.m.put('test.txt', 'This is a test file')
Attachment.m.find().one()
```

Out[2]:
```
{'_id': ObjectId('4f568812eb0330776d000000'),
 'chunkSize': 262144,
 'contentType': u'text/plain',
 'filename': u'test.txt',
 'length': 19,
 'md5': u'0b26e313ed4a7ca6904b0e9369e5b957',
 'uploadDate': datetime.datetime(2012, 3, 6, 21, 56, 34, 255000)}
```

Note here a couple of things. First, `Attachment` seems to be usable as a `ming.collection`, and in fact it is. `Attachment` is actually the collection `forum.attachment.files` (the `forum.attachment.chunks` collection is not accessible as a Ming collection, as you will typically not want to modify it directly.

Second, note that Ming has added a number of extra fields to our document even though we didn't specify them in our `fs.filesystem` definition. These are the fields required to make the gridfs module work properly with the collection. Now let's try looking at the file via the dedicated grifs accessors on the `m` manager:

```
In [3]: a = Attachment.m.get_last_version('test.txt')
        type(a)

Out[3]: gridfs.grid_file.GridOut
```

Ming provides convenience methods to create standard `gridfs GridOut` and `GridIn` instances from PyMongo. Once we have them (as in the `get_last_version` example above), we can use them as any other `GridOut` instance:

```
In [4]: a.seek(0)
        a.read(15)

Out[4]: 'This is a test '

In [5]: a.readline()

Out[5]: 'file'
```

Adding metadata

So once again we appear to have added a layer of complexity with little return. And once again our return comes from the availability of Ming validation. It turns out we can add extra fields and indexes to our `fs.filesystem` just as we did with collections. Suppose we want to store the author of a particular attachment:

```
In [6]: assert Attachment.m.find().count() == 1

        Attachment = fs.filesystem(
            'forum.attachment', sess,
            ming.Field('author', str))
        a = Attachment.m.find().one()
        a.author = 'Rick'
        a.m.save()
        Attachment.m.find().all()

Out[6]: [{'_id': ObjectId('4f568812eb0330776d000000'),
          'author': u'Rick',
          'chunkSize': 262144,
          'contentType': u'text/plain',
          'filename': u'test.txt',
          'length': 19,
          'md5': u'0b26e313ed4a7ca6904b0e9369e5b957',
  'uploadDate': datetime.datetime(2012, 3, 6, 21, 56, 34, 255000)}]
```

Now in order to future-proof our code, we really should move `author` under the `metadata` property:

In [7]:
```
Attachment = fs.filesystem(
    'forum.attachment', sess,
    ming.Field('metadata', dict(
            author=str)))

# by default, ming strips unknown fields coming from the DB
a = Attachment.m.find().one()
a.metadata.author = 'Rick'
a.m.save()
Attachment.m.find().all()
```

Out[7]:
```
[{'_id': ObjectId('4f568812eb0330776d000000'),
  'chunkSize': 262144,
  'contentType': u'text/plain',
  'filename': u'test.txt',
  'length': 19,
  'md5': u'0b26e313ed4a7ca6904b0e9369e5b957',
  'metadata': {'author': u'Rick'},
  'uploadDate': datetime.datetime(2012, 3, 6, 21, 56, 34, 255000)}]
```

Exercises

I. Create a Ming definition of an image filesystem that stores resolution and other metadata.
II. Update the attachment model so that it has "foreign keys" into the `forum.post` table so we can use it for attachments to posts.

In [7]: