Chapter 1

Mapless

Obscenely simple persistence for Smalltalk. Mapless is a small framework for storing objects in a key -> data fashion (i.e., noSQL databases) without requiring any kind of object-data map. So far only MongoDB is supported. And Redis can be used for reactivity and cache joy. Mapless is developed by Sebastian Sastre and the code project is located at: https://github.com/sebastianconcept/Mapless

1.1 Motivation

"I wanted to persist objects with low friction and low maintenance but high scaling and availability capabilities so Mapless is totally biased towards that. This framework is what I came up with after incorporating my experience with Aggregate in airflowing." Sebastian

There is no instVars. There is no accessors. There is no object-mapping impedance. Only persistence.

1.2 Loading it in your image

Take your open image and click for a menu, then tools, then Configuration Browser and search for Mapless. Click Install Stable Version

Or use this snippet to load it into your Pharo image:

```
Gofer it
   smalltalkhubUser: 'Pharo'
   project: 'MetaRepoForPharo30';
   package: 'ConfigurationOfMapless';
   load.
```

```
(Smalltalk at: #ConfigurationOfMapless) load
```

1.3 How does it look?

You can store and retrieve your Mapless models on MongoDB like a breeze. Here is a couple of snippets used while developing tasks (http://tasks.flowingconcept.com/service/)

Getting the connection

```
odb := MongoPool instance databaseAt: 'Reactive'.
```

Create a new model

Just subclass MaplessModel. Here we use a RTask model:

```
task := RTask new description: 'Try Mapless'; beIncomplete.
```

Save it

In Mapless you do things using a do: closure which Mapless uses to automatically discern which MongoDB database and collection has to be used. It also will know if it needs to do an insert or an update. As a bonus, you get the collection and the database created lazily.

Want to save something? Zero bureaucracy, just tell that to the model:

```
odb do:[task save].
```

At http://www.airflowing.com we call this the low-friction way to do it.

Fetching stuff

Getting all models of a given class:

```
odb do:[RTask findAll].
```

Getting a specific model:

```
odb do:[RTask findAt: 'c472099f-79b9-8ea2-d61f-e5df34b3ed06'].
```

For getting efficiently a (sub)group of models, you write your own Mapless class side getters. They should act on the MongoDB indices with the parameters of your query. You'll get your models in a breeze:

```
odb do:[RTask findAtUsername: 'awesomeguy'].
```

Adding something to a model

You can use Mapless models pretending they are prototypical like in Self or Javascript objects. No instVars. No setters. No getters. All just works:

```
odb do:[
   task
      deadline: Date tomorrow;
      notes: 'Wonder how it feels to use this thing, hmm...';
      save].
```

1.4 Need composition?

Of course you do! The only thing you need is to save the children first

```
odb do:[ |anAlert|
  anAlert := RAlert new
             duration: 24 hours;
             message: 'You will miss the target!';
             save.
   task
      alert: anAlert;
      save].
```

This is what we call low-maintenance.

Navigating the object graph

Mapless embraces an aggressive lazy approach. When you query for models you get them. But if they are composed by other (sub)Mapless, they are going to be references and only reify into the proper Mapless model if you send them a message and you can do this with arbitrary depth:

```
odb do:[task alert message].

odb do:[task alert class = MaplessReference].  "<- true"

odb do:[task alert description].  "<- 'You will miss the target!'"
```

1.5 Persisting a different model

So you now need to store a different kind of models, say List or User or anything, how you proceed?

This is what happens to you with Mapless:

- Create a subclass for them
- Know what attributes they will need in advance
- Create its attributes' instVars
- Make accessors for them
- Elegantly map them so it all fits in the database
- Patiently re-map them every time you need to change its design
- Profit

How does it look? on Redis

Redis is interesting for:

- Caching. It will hold the data in RAM so you get great response times.
- Reactivity. Mapless uses Redis PUB/SUB feature to observe/react models among N Pharo worker images enabling horizontal scaling.

Here is a workspace for Redis based Mapless models:

```
redis := RedisPool instance.
guy := MaplessRedisDummyPerson new
          firstName: 'John';
          lastName: 'Q';
          yourself.
redis do:[:c| guy save].
redis do:[:c| MaplessRedisDummyPerson findAt: '38skolpqv0y9lazmk772hmsed'].
redis do:[:c| (MaplessRedisDummyPerson findAt: '38skolpqv0y9lazmk772hmsed')
      lastName].
```

1.6 State and Future

This code is considered alpha. Check its tests. Contribute!

Contributions are very welcomed, send that push request and hopefully we can review it together.

Direction?

We would love to see more and better tests and iterate the reactive features so you can ultimately get an a model in one image being observed in regard to one event by something in another image and that something reacting upon that event. Broadcast and multicast of events would be also a nice feat and not too hard to do. Have design suggestions? Let's have a chat!

For MongoDB-based Mapless, a nice feature would be to have UserModel ensureIndex: '{ key1: 1, key2: -1 }' We actually are starting to think Amber and localStorage but that's more hush hush at this point. Oh gee! we already blown it!

Remember to get Pharo Smalltalk it is as easy as running in your terminal:

```
wget -O- get.pharo.org/30+vm | bash
```