Developing applications with Smalltalk and CouchDB

J. Marí Aguirre

A practical guide to develop CouchDB Applications on Smalltalk

Developing applications with Smalltalk and CouchDB

Introduction

I started learning CouchDB because I wanted to develop an idea of a friend of mine. When I read that it's possible to design a view server on any language, I thought it would be great to have one in Smalltalk, Smalltalk in all of its versions has a great debugging environment, in the other hand CouchDB lacks on debugging capabilities. When I began to delve into the subject I realized that just a view server would not be enough to develop a real application in any language, that's the reason to study the complete view server javascript source code and port it to Smalltalk as well as I could, I'm not precisely a Smalltalk skilled programmer nor an expert in Javascript, to be precise I'm not specialized in any particular environment, but the fact is that it works and It's a great project that give us the possibility to deploy real CouchDB applications in Smalltalk as powerful as you can do in javascript. After codding Shows and Lists, updates…, and all the CouchDB tricks, I developed a couchapp'like environments to be able to upload the application from the Smalltalk Image to CouchDB, I needed Templates and Evently too… Real web applications without templates could be a mess although you develop the application in Seaside, and I decided to port Mustache to Smalltalk from the javascript source code.

Evently is an important piece in this puzzle and I added the possibility to create an evently application residing in the Smalltalk image.

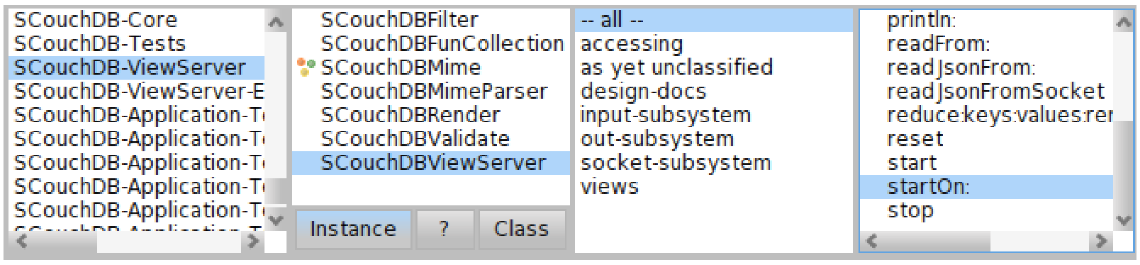
I hope you enjoy SCouchDBViewServer! I suggest you to follow the jQuery mobile + CouchDB series by Todd Anderson from http://custardbelly.com/blog/2010/12/08/jquery-mobile-CouchDB-part-1-getting-started/. This document is not enough to learn jQuery mobile, the purpose of this document is to learn how to develop CouchDB applications in Smalltalk.

We'll need a fresh Pharo 1.3 Image with the SCouchServer package installed. Of course CouchDB from http://CouchDB.apache.org, and Socat, multipurpose relay application from http://www.dest-unreach.org.

Socat is a utility similar to Netcat, despite the fact Netcat is a more standard tool, Socat has the invaluable –s option to ignore errors when CouchDB breaks the pipe with him. Socat is the glue to join CouchDB with Smalltalk.

SCouchServer uses a standard socket instead the standard IO because Sockets are a more natural way to communicate the Smalltalk Image out of its world, Standard IO is recently adopted on Pharo but, due to its interactive nature, use the Standard IO is not a good solution.

The use of sockets gives us the possibility to install the view server in a different host than the CouchDB host or run multiple instances of the ViewServer listening on different ports.



Browser view of the SCouchDB-ViewServer package

SCouchDBViewServer Installation

You can find the needed packages at squeak source 3 site http://ss3.gemstone.com.ss/SCouchDBViewServer.html, load all the packages using Montichello in the following order:

JSON
SCouchDB-Core
Mustache
SCouchDB-ViewServer
ScouchDB-Application
SCouchDB-Test

If you want to experiment with the JQuery plugin for seaside you have to load the SCouch-JQuery package.

If you want to run the view server on Squeak you will need JSON-Squeak and SCouchDB-Core-Squeak

Setting up the environment

CouchDB needs to know witch executable file has to run to start up the server. Just write a new query server field inside the configuration file /etc/CouchDB/local.ini. We will add the following line in the section [query_servers].

```
smalltalk = "the\complete\path\to\socat -s - TCP:127.0.0.1:8181"
```

With this line, CouchDB will be able to create a new Socat process, as we said in the previous chapter, Socat connects the standard input/output/error to a standard Socket, the option –s makes Socat fault tolerant, we need –s option because CouchDB creates a ViewServer process and the pipes to it, and after the creation CouchDB gets its PID. Other second process uses this PID to locate the process and rebuild the pipe, this new process will be responsible of the communication and the first one dies, breaking the first pipe, causing an error in Socat, fortunately we can ignore the error using –s option.

Creating our first Database

Once our environment is up and running, we are ready to create our first database using the Smalltalk SCouchDB and add a new document in our new database.

```
|aNewDoc db adaptor|
aNewDoc:=JsonObject new.
aNewDoc at: 'type' put:'person'.
aNewDoc at:'name' put:'John'.
aNewDoc at:'job' put:'Software Enginiering'.
adaptor:=SCouchDBAdaptor new
	host: '127.0.0.1'
	port: 5984
	userName:'admin'
	password:'******'.
db:= adaptor ensureDatabase: 'smalltest'.
db documentPut: ( aNewDoc ).
```

We can retrieve all documents from the database using the SCouchDatabase #alldocuments method.

```
|docArray|
...
docArray:=db allDocuments.
docArray do:[:aDoc|
	Transcript show: aDoc asString.
]
...
Transcript
------------
a JsonObject('_id'->'2493d7307ae7a9cb3af0bca0b8002f6f' '_rev'->'1-
3fbf72fa259931f56afe8e51e7c4d39a' 'job'->'Software Enginiering' 'name'->'John' 'type'->'person' )
```

SCouchDBAdaptor has two methods to create a database, #ensureDatabase: aDatabase and #createDatabase: aDatabase, the first one attempts to create a new database, and if the database exists returns the reference to the existing database, on the other hand #createDatabase will fail if the database exists.

You can check the existence of a database using #hasDatabase: aDatabase and delete it with #deleteDatabase: aDatabase.

SCouchDBAdaptor

#ensureDatabase: aDatabase	Attempts to create a new database, and if the database exists returns the reference to the existing database
#createDatabase: aDatabase	Attempts to create a new database, and if the database exists raises an error
#hasDatabase: aDatabase	Checks if a database exists.
#deleteDatabase: aDatabase	Deletes a database
#host: aHost port: aPort userName:aUserName password:aPassword	Logs in to CouchDB
#logout	Closes the current connection
#session	Logs in to CouchDB using the previous assigned host, port, username and password accesors.
#users	Retrieve the users database
#activeTasks	
#config	Reads the server configuration
#databases	Gets an array of the databases in the server
#uuids	

Playing with Documents

There are two properties required in any json documents stored in CouchDB, _id and _rev property. The first one is assigned by CouchDB unless you assign a value to the _id property. Normally you don't want to deal with _id's despite the fact that you have to store Design Documents. The other required property is _rev, as the _id property, _rev will be set by CouchDB unless you put a value in.

If you put the same object again using #documentPut: aDoc, a new Document will be created with a different _id and _rev, therefore if what you need to do is modify an existing document you must read the document from the database and write it again

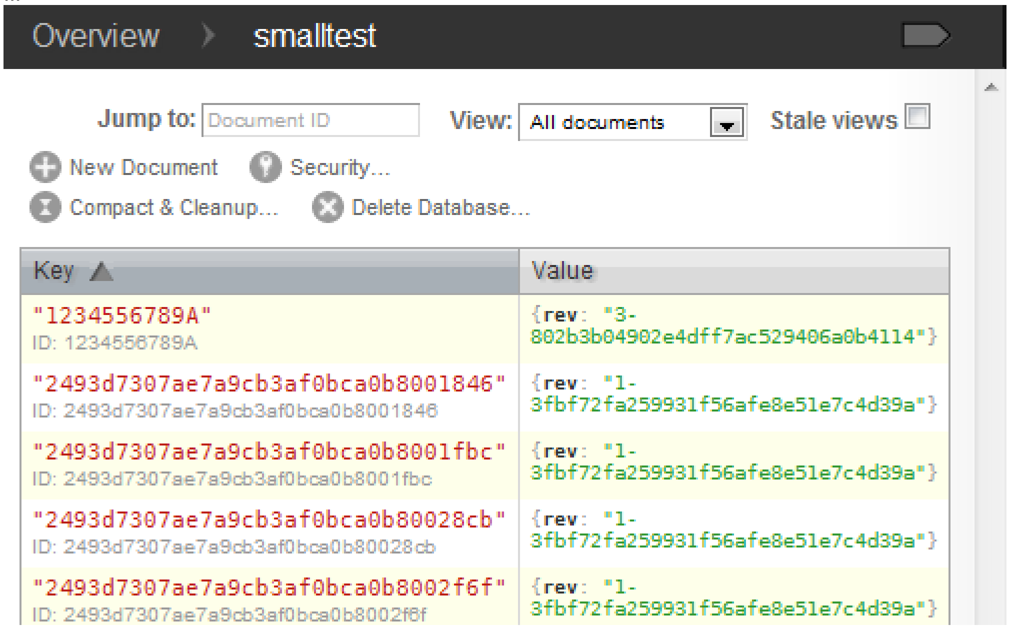
The only available method to get documents from CouchDB without specifying the _id field is allDocuments, of course that retrieving all the database documents is crazy in a real application, that's the reasons we need to learn CouchDB views. Any way you can assign an _id to your jsonObject and put the document using #documentPut: or #documentAt: id put: jsonObject.

```
...
aNewDoc _id:'1234556789A'.
```

```
...
adaptor:=SCouchDBAdaptor new
        host: '127.0.0.1'
        port: 5984
        userName:'admin'
        password:'******'.
db:= adaptor ensureDatabase: 'smalltest'.
db documentPut: ( aNewDoc ).
```

Now the document is stored with the assigned _id property, and if we try to write it again an error will be raised because the object already exists, and CouchDB prevents to overwrite a document unless you don't specify its latest revision. The main reason for this behavior is simply to maintain the consistency of a document against simultaneous changes. The method #documentAt:id put:aJsonObject takes care of it and if a revision error occurs, the method will retrieve the last revision property and attempts to update the document again. Be careful with this method because it breaks the consistency protection of CouchDB. Now we are just going to change the way we put the document in the database and we will observe the difference using Futon interface.

```
...
aNewDoc at:'name' put:'Peter'.
db documentAt:'1234556789A' put: ( aNewDoc ).
aNewDoc at:'name' put:'Charles'.
db documentAt:'1234556789A' put: ( aNewDoc ).
...
```



Key	Value
"1234556789A" ID: 1234556789A	{rev: "3-802b3b04902e4dff7ac529406a0b4114"}
"2493d7307ae7a9cb3af0bca0b8001846" ID: 2493d7307ae7a9cb3af0bca0b8001846	{rev: "1-3fbf72fa259931f56afe8e51e7c4d39a"}
"2493d7307ae7a9cb3af0bca0b8001fbc" ID: 2493d7307ae7a9cb3af0bca0b8001fbc	{rev: "1-3fbf72fa259931f56afe8e51e7c4d39a"}
"2493d7307ae7a9cb3af0bca0b80028cb" ID: 2493d7307ae7a9cb3af0bca0b80028cb	{rev: "1-3fbf72fa259931f56afe8e51e7c4d39a"}
"2493d7307ae7a9cb3af0bca0b8002f6f" ID: 2493d7307ae7a9cb3af0bca0b8002f6f	{rev: "1-3fbf72fa259931f56afe8e51e7c4d39a"}

Documents in Futon

As we expect the "1234556789A" object has three versions, if you want to delete all of them you should know the last revision and send the #deleteDocument: id revision:rev method. You can use #deleteDocument:id instead, it'll retrieve the last revision for you.

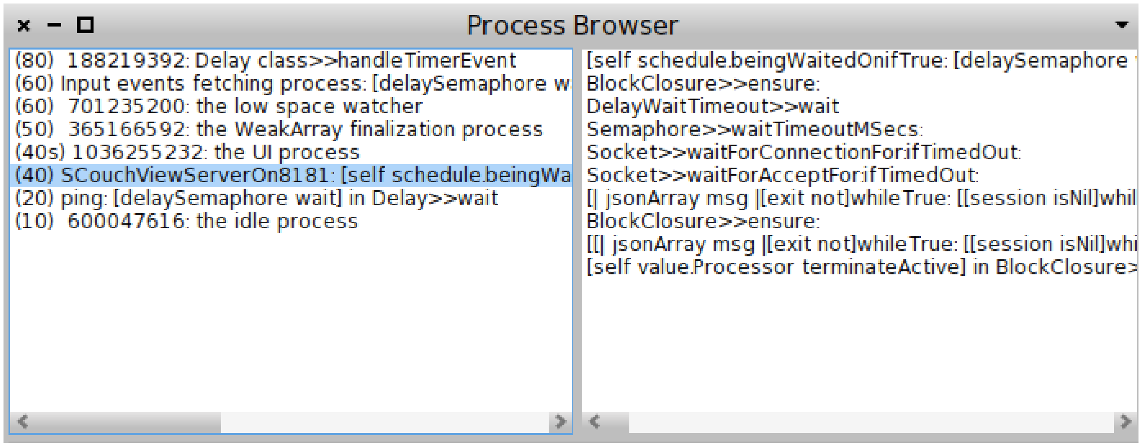
#deleteDocument: id	Deletes a document given an _id
#deleteDocument: id revision: aRev	Deletes a document given an _id and its last revision.
#documentAt: id	reads a document given its _id.
#documentAt: id ifAbsent: aBlock	reads a document given its _id if the document is absent evaluates the blockClosure.
#documentAt: id ifAbsentPut: aBlockOfJsonObject	reads a document given its _id if the document is absent evaluates the BlockClosure this block must assign a JsonObject in its last statement.
#documentAt:id put: aJsonObject	writes a document given its _id and the JsonObject.
#documentAt:id rev: aJsonObject	reads a document given its _id and the revision.
#documentCurrentRevAt:id	reads the last revision of a Document given its _id property.
#documentPut: aJsonObject	writes a document, it will retrieve a new _id for the new document.
#documentRevisionsAt: id	reads the document revisions array, and the number of the last one.
#documentRevsInfoAt: id	reads the document revisions array with the status.

Our First Application. Album database

We are going to guide our learning on the great jQuery mobile tutorial by Todd Anderson from his blog http://custardbelly.com/blog/2010/12/08/jquery-mobile-CouchDB-part-1-getting-started/. After all, be sure the ViewServer is running, start it doing:

```
SCouchDBViewServer start[alt+d]
```

Open the process Browser and check if there is a SCouchViewServer process running



```
(80) 188219392: Delay class>>handleTimerEvent
(60) Input events fetching process: [delaySemaphore w
(60) 701235200: the low space watcher
(50) 365166592: the WeakArray finalization process
(40s) 1036255232: the UI process
(40) SCouchViewServerOn8181: [self schedule.beingWa
(20) ping: [delaySemaphore wait] in Delay>>wait
(10) 600047616: the idle process
```

```
[self schedule.beingWaitedOnifTrue: [delaySemaphore
BlockClosure>>ensure:
DelayWaitTimeout>>wait
Semaphore>>waitTimeoutMSecs:
Socket>>waitForConnectionFor:ifTimedOut:
Socket>>waitForAcceptFor:ifTimedOut:
[| jsonArray msg |[exit not]whileTrue: [[session isNil]whil
BlockClosure>>ensure:
[[| jsonArray msg |[exit not]whileTrue: [[session isNil]whi
[self value.Processor terminateActive] in BlockClosure>
```

SCouchViewServer listening on port 8181

The default port is 8181, you can start the ViewServer on a different port sending the message #startOn: aPort. Having two or more ViewServers running on the same image is also possible.

Let’s create a database called ‘albums’ with two documents.

```
|docs db adaptor|

 docs:= Array
with:(JsonObject fromAssociations:{    'artist'->'Thin Lizzy' .
                                   'title'->'Fighting'.
                                   'description'->'Bona-fide Rock classic'
})
with:(JsonObject fromAssociations:  {        'artist'->'David Bowie' .
                                   'title'->'Honky Dory'.
'description'->'Doesn''t get much better than this'                          }) .


adaptor:=SCouchDBAdaptor new
      host: '127.0.0.1'
      port: 5984
      userName:'admin'
      password:'********'.
db:= adaptor ensureDatabase: 'albums'.


docs do:[:d|
      db documentPut: d.
      ]
```

Let’s create now our CouchDB application, we need a new subclass from SCouchDBApp class

```
SCouchDBApp subclass: #Albums
      instanceVariableNames: ''
      classVariableNames: ''
      poolDictionaries: ''
      category: 'SCouchDB-Application-Test'
```

A good way to prepare our application is the #initialize method, where we can declare our views, shows and any other CouchDB component.

SCouchDBApp class is a JsonObject subclass (JsonObject→SCouchDBAppWrapper→SCouchDBApp), as in CouchDB, SCouchDBApp is a json document with some especial methods that will help you in your design procedure.

First of all let’s assign the name and the language of our application design document, this could be done with the #appname: aName and #language:aLanguage methods, anyway if you don’t assign any value, class name in lower case and ‘smalltalk’ will be the default values.

```
Albums initialize
self appName: 'albums'.
self language: 'smalltalk'
self initViews.
…
```

For a better organization of our code we are going to create an #initViews method where we’ll place our views code. Views are implemented using smalltalk BlockClosure, as in javascript unnamed functions, you can declare a block of code and assign it to a variable name, or directly store it inside your couch application.

Views always receive one parameter, a Document, the view method is evaluated for all the documents stored in the database, except the design documents, CouchDd will create an internal b-tree storing in all the emitted documents indexed by a key.

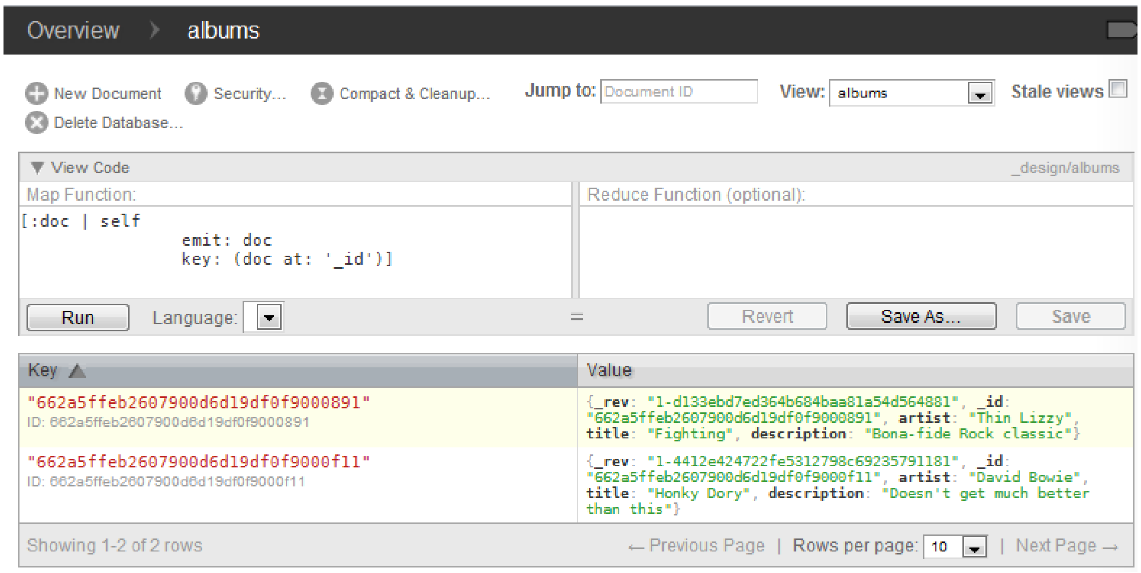
```
Albums initViews.
self addViewAt:'albums'
        put:
        [:doc|
                self emit: doc key:(doc at:'_id')
        ].
```

This simple view will emit all the documents and its _id as key.

Now let's try our Application, the following step is upload the application to CouchDB.

```
Albums initialize
self appName: 'albums'.       "you can omit this"
self language: 'smalltalk'    "you can omit this"
self initViews.
self connectTo:'localhost' port:5984 userName:'admin' password:'********'.
self upload.
```

```
Workspace
        Albums new [alt+d]
```



```
[:doc | self
            emit: doc
            key: (doc at: '_id')]
```

Key	Value
"662a5ffeb2607900d6d19df0f9000891" ID: 662a5ffeb2607900d6d19df0f9000891	{_rev: "1-d133ebd7ed364b684baa81a54d564881", _id: "662a5ffeb2607900d6d19df0f9000891", artist: "Thin Lizzy", title: "Fighting", description: "Bona-fide Rock classic"}
"662a5ffeb2607900d6d19df0f9000f11" ID: 662a5ffeb2607900d6d19df0f9000f11	{_rev: "1-4412e424722fe5312798c69235791181", _id: "662a5ffeb2607900d6d19df0f9000f11", artist: "David Bowie", title: "Honky Dory", description: "Doesn't get much better than this"}

Albums View in futon.

You can edit a View with Futon and get the result, it's very useful to check your views and the emitted results, don't forget to copy the code to your view in the Smalltalk Application if you modify it.

Futon seems to have a bug in the language selector, and you won't be able to select the language until you edit and execute a view stored in a design document.

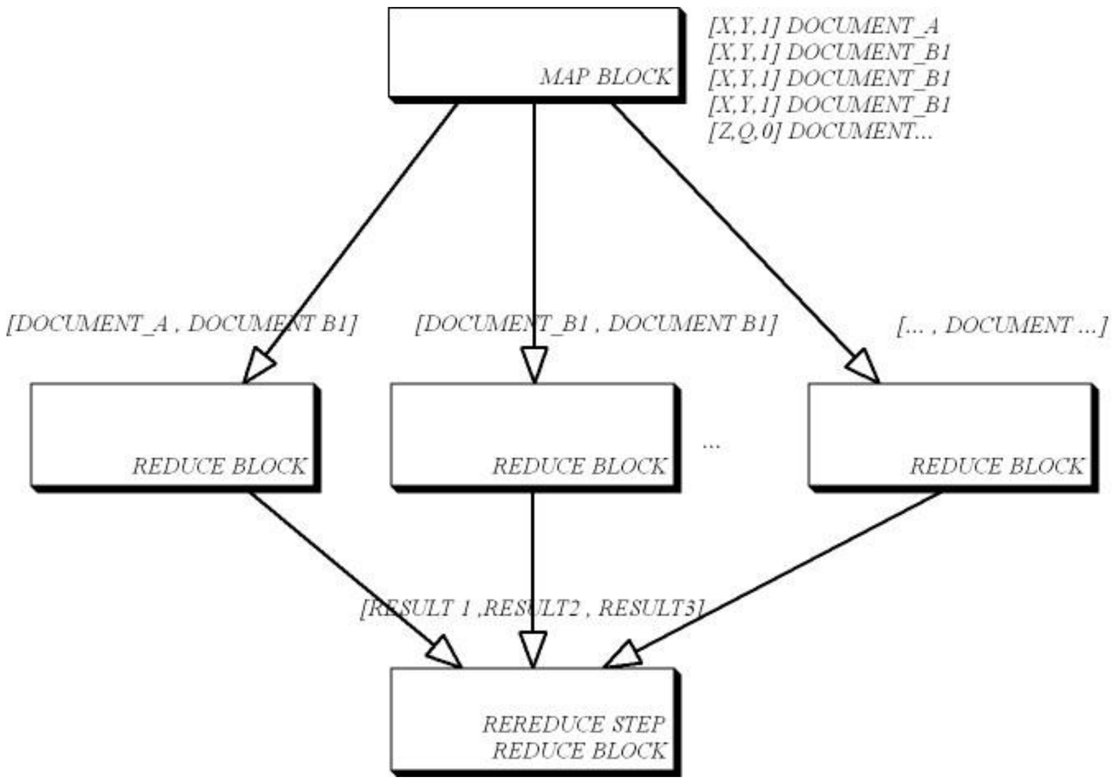
Understanding Map-reduce views

MapReduce Views are blocks that will transform in several steps, an array with all of your documents stored in your database to a new array offering only the information you want to emit. CouchDB caches the result, that means the View is executed the first time for every Document and only will be executed again if there are changes in the database.

In the Map step those documents will be sent one by one as the first parameter to each Map block. The View Server will respond with an array of arrays of key/value pairs resulting from the Map execution. [[(m1:d1), (m2:d1)][(m1:D2),(m2:d2)]…[(m1:dn),(m2:Dn)]].

When a View defines a Reduce block a second step is done, in this step CouchDB sends the Reduce block and a Map block results array (key value pairs) to the View Server, depending on the B-tree node size created by CouchDB will send the results by parts, each time the View Server receives the reduce command, the Server transforms this array in two new arrays a Keys array and a Values array, and then compiles and calls the Reduce Block with those two arrays. After the View server sends back the result to CouchDB for every part of the Map result, CouchDB ask to the View Server again, this time with the command rereduce and the results of the reduce blocks, rereduce parameter will be set to "true" and the keys array will be nil.

The Boolean parameter called rereduce set to true will helps you to identify this last step where you must recalculate the results for the intermediate results.



Map-Reduce View execution

The following source code implements the **count** build-in reduce function, as you can see in the source, in the reduce step the number of elements is the array **v** size, but inside the rereduce step it will receive an array of all the computed sizes, then the total number of elements will be the sum of all of these "sizes".

```
[:k :v :rereduce|
    |result|
    result:=0.
    rereduce
```

```
	ifTrue:[
		v do:[:e| result:= result + e ].
	]
	ifFalse:[result := v size].
	result "returns result"
 ]
```

Now you can see a more complex example, the **statistics** reduce function.

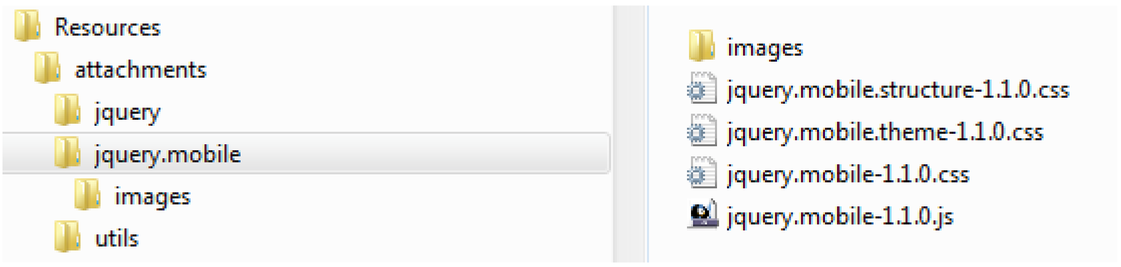
```
[:k :v :rereduce|
	|avg min max sum  count sqSum stdDev variance result|

	avg:=0.min:=0.max:=0.sum:=0.count:=0.stdDev:=0.sqSum:=0.variance:=0.

	rereduce
	ifTrue:[
		min:= (v at:1) min asFloat .
		max:= (v at:1) max asFloat.
		v do:[:e|
			(e min asFloat < min) ifTrue:[min:=e min asFloat].
			(e max asFloat > max) ifTrue:[max:=e max asFloat].
			sum:= sum + e sum asFloat.
			sqSum:= sqSum+ e sumSqr asFloat .
			count:= count + e count
		].
		avg := (sum / count) asFloat.
		variance:=(((sqSum ) - (sum*sum/count )) / (count -1) ).
		stdDev :=variance sqrt.
		result :=JsonObject fromAssociations: {
						'min'-> min.
						'max'-> max.
						'avg'-> avg.
						'sum'-> sum.
						'count'-> count.
						'variance'->variance.
						'stdDev'->stdDev.
						'sumSqr'->sqSum
						}.
	]
	ifFalse:[
		min:= (v at:1) asFloat.
		max:= (v at:1) asFloat.
		sum:= 0.
		count:= v size.
		v do:[:e|
			(e asFloat < min) ifTrue:[min:=e asFloat].
			(e asFloat > max) ifTrue:[max:=e asFloat].
			sum:= sum + e asFloat.
			sqSum:= sqSum+ ((e asFloat )* (e asFloat))
		].
		avg :=( sum / count) asFloat.
		variance:=(((sqSum ) - (sum*sum/count )) / (count -1) ).
		stdDev :=variance sqrt.
		result :=JsonObject fromAssociations: {
						'min'-> min.
						'max'-> max.
						'avg'-> avg.
						'sum'-> sum.
						'count'-> count.
						'variance'->variance.
						'stdDev'->stdDev.
						'sumSqr'->sqSum
						}.
		].
	]
```

Retrieving a View from Smalltalk is as easy as retrieving any other document from the database, the _id you must pass to the #documentAt: message is "_design/designdoc/_view/viewname". If you don't want to get the reduced results and you want the view emit array you should specify the parameter reduce=false, "_design/designdoc/_view/viewname?reduce=false".

The application interface. First approach

Now it's the time to give it an interface! We are going to design our first application interface, so we we'll need to attach files to our design document. CouchDB stores all the static contents as attachments of a document, there is no limit on the number of files you can attach to a document.

Download the latest jQuery mobile and place the js file inside a folder, for instance we will place jQuery files in the resources folder of our Smalltalk image, create a subdir called attachments and copy the following files.



Attachments folder

Create a file called "load.js" and save it inside the attachment folder. In this example we have placed it in the attachment/utils/[loader.js] folder. This file will include all the javascript libraries in our main application page.

```
function couchapp_load(scripts){
    for (var i=1;i<scripts.length;i++){
        document.write('<script src="'+scripts[i]+'"></script>')
    };
};
couchapp_load([          "/_utils/script/json2.js",
                "/_utils/script/sha1.js",
                "./jquery/jquery-1.7.2.min.js",
                "./jquery.mobile/jquery.mobile-1.1.0.js",
                "/_utils/script/jquery.couch.js?0.11.0",
                "/_utils/script/jquery.dialog.js?0.11.0",
                ]);
```

Create an index.html file with the next code, and save the file inside the attachment folder

```
<html>
    <head>
        <title>My Albums</title>
        <link rel="stylesheet" href="style/main.css" type="text/css">
        <link rel="stylesheet" href="./jquery.mobile/jquery.mobile-1.1.0.css" type="text/css">
    </head>
    <body>
    <div data-role="page">
        <div data-role="header"><h2>Albums</h2></div>
        <div data-role="content">
            <ul id="albums" data-role="listview" data-theme="c" data-dividertheme="b"></ul>
        </div>
        <div data-role="footer">Footer</div>
    </div>
    </body>
    <script type="text/javascript" src="./utils/loader.js"></script>
    <script type="text/javascript" charset="utf-8">
        $db = $.couch.db("albums");
        function handleDocumentReady(){
            //request album documents
            refreshAlbums();
        }
        function refreshAlbums(){
            $("#albums").empty();
            $db.view("albums/albums",
            {
            success:function(data){
            var i;
            var album;
            var artist;
            var title;
            var description;
            var listItem;
            for (i in data.rows){
                album = data.rows[i].value;
                artist =album.artist;
                title = album.title;
                description =   album.description;
                listItem ="<li>"        +
                "<h2 class=\"artist\">" + artist +"<\/h2>" +
                "<p class=\"title\">" + title +"<\/p>" +
                "<p class=\"description\">" + description +"<\/p>";
                $("#albums").append(listItem);
            }
            $("#albums").listview("refresh");
        }});
        }
        $(document).ready(handleDocumentReady);
    </script>
</html>
```

The last step is to import the entire attachments folder in our Album application. In your Smalltalk Album #initialize code append the import line and upload the application to CouchDB.

```
initialize
self appName: 'albums'.
self language: 'smalltalk'.
self initViews.
self importAllAttachmentsFrom:
'D:\smalltalk\SCouchViewServer\Contents\Resources\attachments\'.
self connectTo:'localhost' port:5984 userName:'admin' password:'********'.
self upload.

Workspace
    Albums new [alt+d]
```

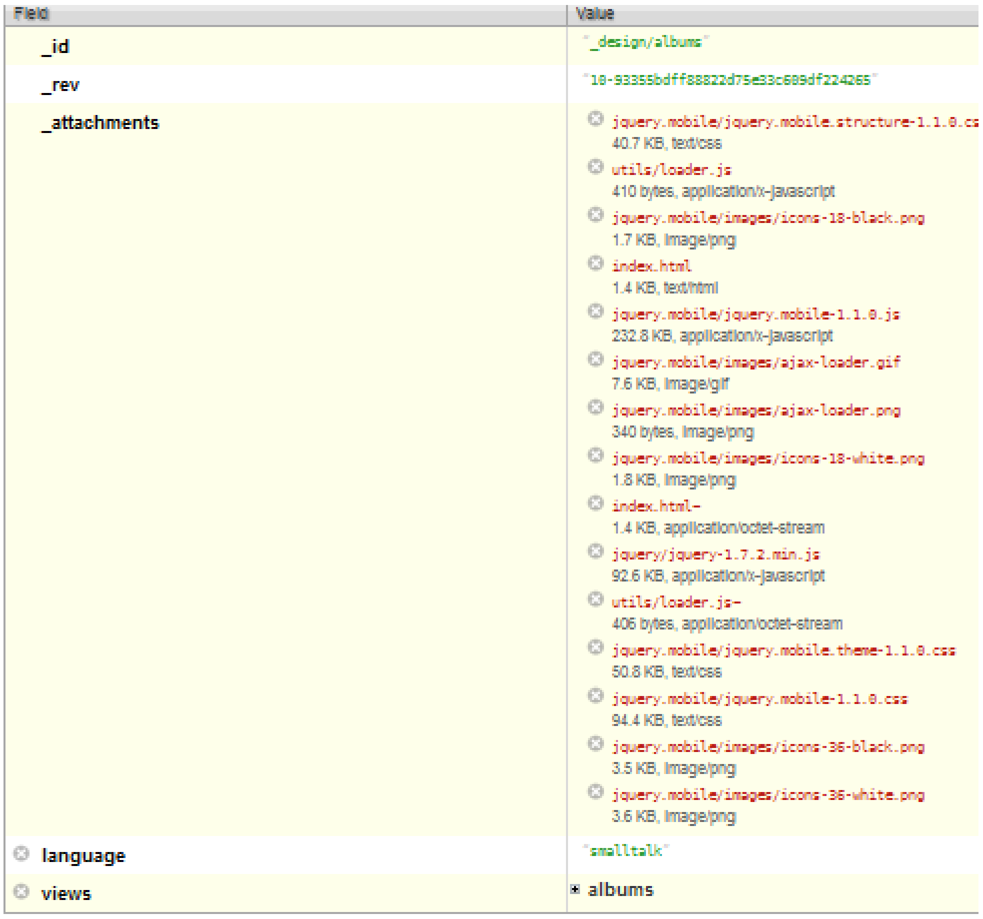
Now the Albums design document must look like the following image. Take a look to the attachment property, in CouchDB an attachment can be accessed with the url pattern …/batabaseName/_design/designDocument/attachmentName.

http://127.0.0.1:5984/albums/_design/albums/index.html

If the attachment is placed in a subfolder, you can download it encoding de / character as %2

http://127.0.0.1:5984/albums/_design/albums/utils%2floader.js

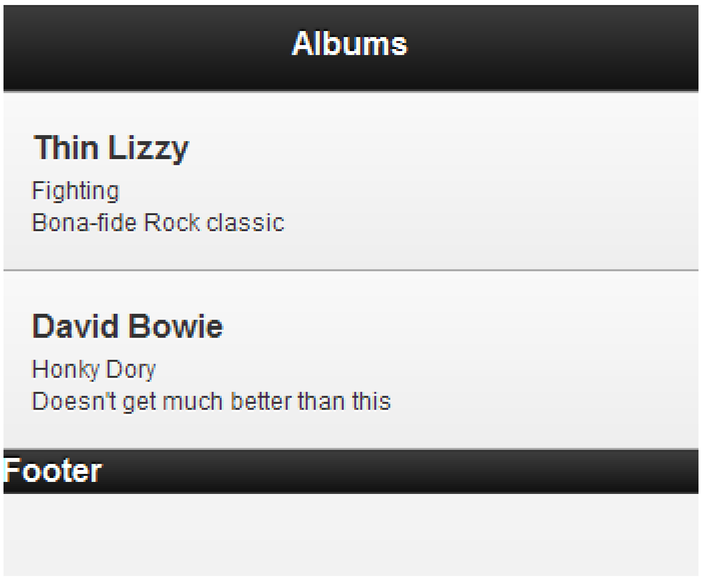
http://127.0.0.1:5984/albums/_design/albums/jquery.mobile%2fimages%2ficons-18-black.png



Field	Value
_id	"_design/albums"
_rev	"10-93355bdff88822d75e33c609df224265"
_attachments	jquery.mobile/jquery.mobile.structure-1.1.0.cs 40.7 KB, text/css utils/loader.js 410 bytes, application/x-javascript jquery.mobile/images/icons-18-black.png 1.7 KB, Image/png index.html 1.4 KB, text/html jquery.mobile/jquery.mobile-1.1.0.js 232.8 KB, application/x-javascript jquery.mobile/images/ajax-loader.gif 7.6 KB, Image/gif jquery.mobile/images/ajax-loader.png 340 bytes, Image/png jquery.mobile/images/icons-18-white.png 1.8 KB, Image/png index.html~ 1.4 KB, application/octet-stream jquery/jquery-1.7.2.min.js 92.6 KB, application/x-javascript utils/loader.js~ 406 bytes, application/octet-stream jquery.mobile/jquery.mobile.theme-1.1.0.css 50.8 KB, text/css jquery.mobile/jquery.mobile-1.1.0.css 94.4 KB, text/css jquery.mobile/images/icons-36-black.png 3.5 KB, Image/png jquery.mobile/images/icons-36-white.png 3.6 KB, Image/png
language	"smalltalk"
views	albums

Albums design document

Open your favorite browser and browse to
http://127.0.0.1:5984/albums/_design/albums/index.html



Albums

Thin Lizzy
Fighting
Bona-fide Rock classic

David Bowie
Honky Dory
Doesn't get much better than this

Footer

Albums first interface.

Normally in every application we can see a general list with all of the elements stored in the database, and when we click on one of them, the page is redirected to a new page that shows us a detailed view of the selected element.

The page concept is clearly adopted by jQuery mobile using the data-role=”page” property on a <div> element, a page can be declared inside the same html document or just include an external reference to the external file. By now, we’ll use de first approach and we’ll adopt the second one in following sections.

Let’s open the Index.html file and add the next code after the first <div> page. This code adds a second page with the detailed show of an Album.

```
…
<div data-role="page" id="albumview">
            <div data-role="header"><h2 class="albumtitle"></h2></div>
            <div data-role="content" id="albumcontent"></div>
            <div data-role="footer">Footer</div>
</div>
</body>
…
```

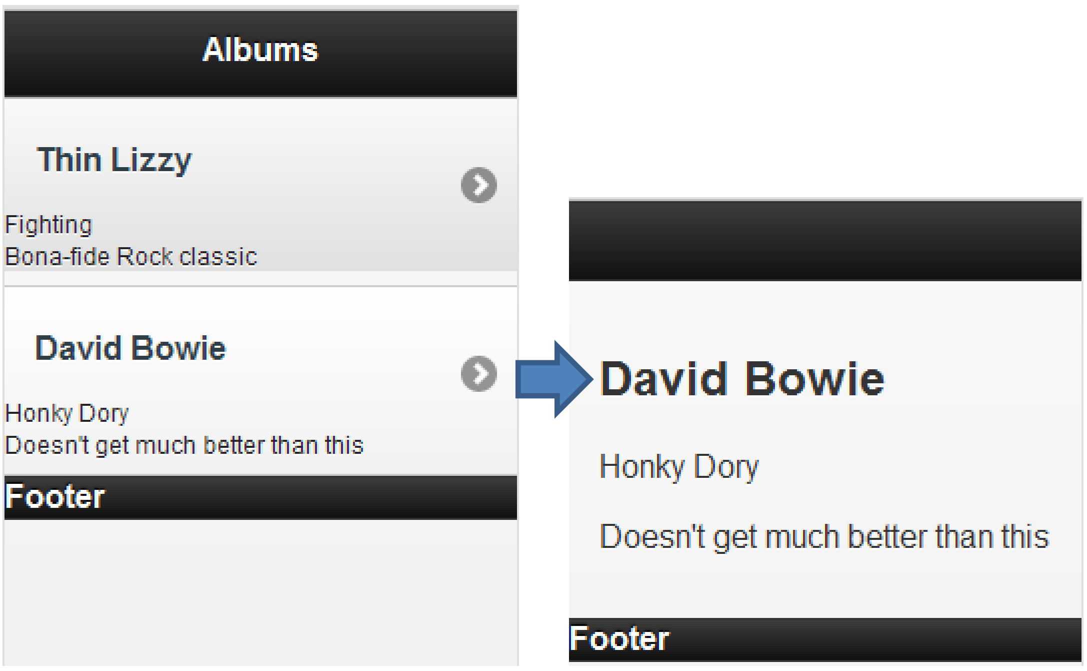
We need to add the link to our second page in every album of the Albums list. Inside the album list for..next change the content…

```
…
listItem = $("<li/>" , { class:"album"});
header = "<h2 class=\"artist\">" + artist +"<\/h2>";
albumLink= $("<a/>",{
            href: "#albumview",
            "data-identity":album._id,
            click: function(){albumId=$(this).data("identity");}
            });
albumLink.append(header);
listItem.append(albumLink);
listItem.append("<p class=\"title\">" + title +"<\/p>");
listItem.append("<p class=\"description\">" + description +"<\/p>");
$("#albums").append(listItem);
…
```

We need a function to load the document from the database and append the data into the second page. This function will be called by the “pagebeforeshow” event included inside the handleDocumentReady function.

```
…
$("#albumview").bind("pagebeforeshow",openAlbum);
…

function openAlbum(){
      $("#albumcontent").children().remove();
      $db.openDoc(albumId,
            {
            success: function (data){
                  var artist=data.artist;
                  var title= data.title;
                  var description = data.description;
                  var html="<h2 class=\"artist\">"+artist+"</h2>"+
                              "<p class=\"title\">"+ title + "</p>" +
                              "<p class=\"description\">"+ description + "</p>";
                  $("#albumcontent").append(html);
                  $("#albumheader .albumtitle").text(title);
                  }
            });
}
```

Deploy the application and test it.

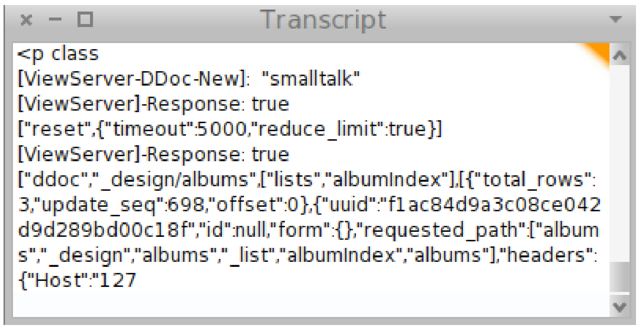


Document detailed

Debugging applications

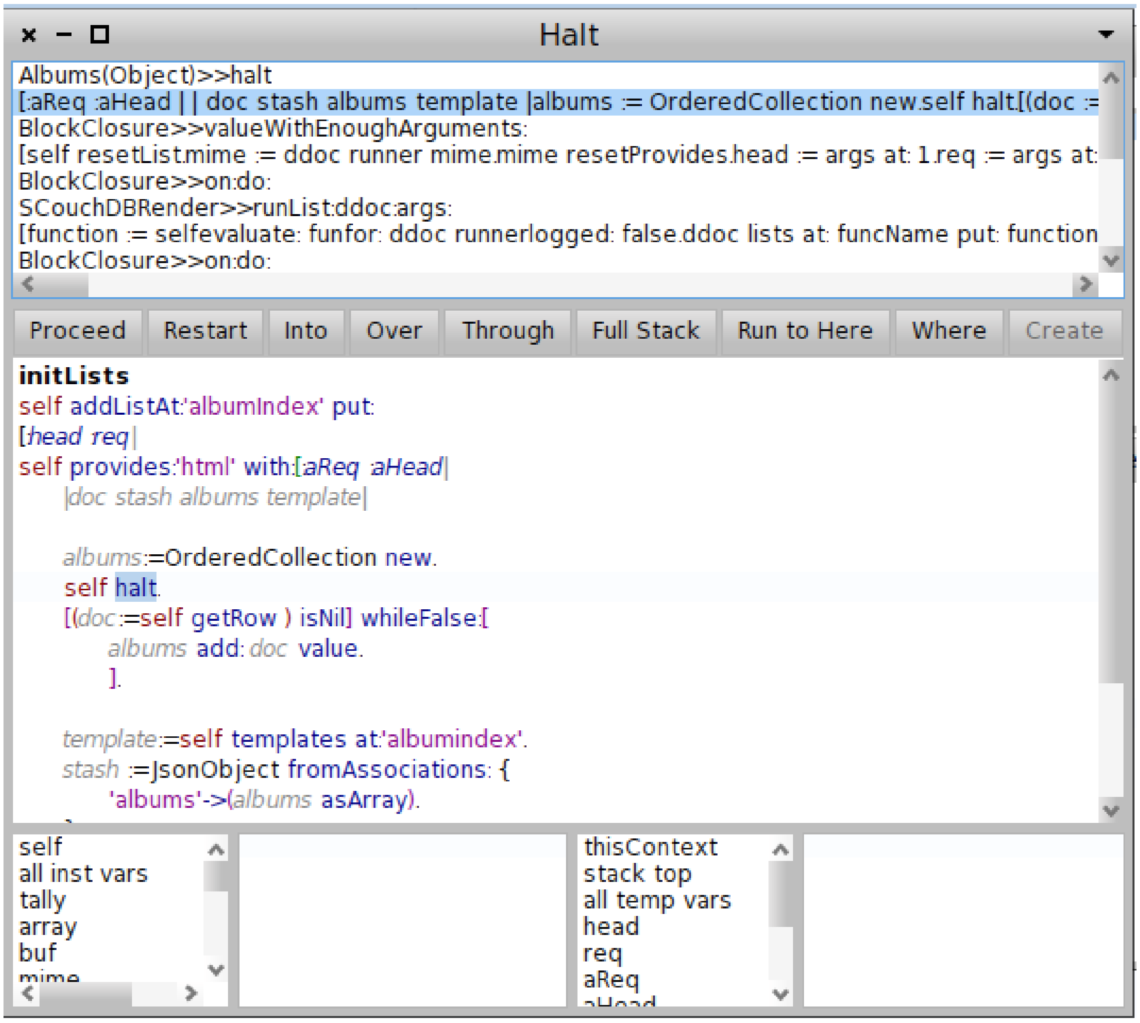
You can activate debug option sending #debugOn or #debugOff to SCouchDBViewServer class, a lot of information will be printed on Transcript, and written into a file called ViewServer-debug.txt placed in the resources folder.

A good way to debug a View, List … or any other kind of BlockClosures is sending the self #halt message to bring up the debugger window and check the block behavior. Don't forget to upload the application again, although our source code is updated we need to update our design document in the database and in the ViewServer application cache.



```
<p class
[ViewServer-DDoc-New]:  "smalltalk"
[ViewServer]-Response: true
["reset",{"timeout":5000,"reduce_limit":true}]
[ViewServer]-Response: true
["ddoc","_design/albums",["lists","albumIndex"],[{"total_rows":
3,"update_seq":698,"offset":0},{"uuid":"f1ac84d9a3c08ce042
d9d289bd00c18f","id":null,"form":{},"requested_path":["album
s","_design","albums","_list","albumIndex","albums"],"headers":
{"Host":"127
```

SCouchDBViewServer debugOn, Transcript



```
Albums(Object)>>halt
[:aReq :aHead | | doc stash albums template |albums := OrderedCollection new.self halt.[(doc :=
BlockClosure>>valueWithEnoughArguments:
[self resetList.mime := ddoc runner mime.mime resetProvides.head := args at: 1.req := args at:
BlockClosure>>on:do:
SCouchDBRender>>runList:ddoc:args:
[function := selfevaluate: funfor: ddoc runnerlogged: false.ddoc lists at: funcName put: function
BlockClosure>>on:do:
```

```
initLists
self addListAt:'albumIndex' put:
[head req|
self provides:'html' with:[:aReq :aHead|
    |doc stash albums template|

    albums:=OrderedCollection new.
    self halt.
    [(doc :=self getRow ) isNil] whileFalse:[
        albums add: doc value.
        ].

    template:=self templates at:'albumindex'.
    stash :=JsonObject fromAssociations: {
        'albums'->(albums asArray).
```

```
self            thisContext
all inst vars   stack top
tally           all temp vars
array           head
buf             req
mime            aReq
                aHead
```

Debugging albumIndex blockClosure list

Shows in Smalltalk

A show defines how a document will look in a client application.

Shows, as Views, are implemented with BlockClosures, a show block receives two arguments, aDoc and the request JSonObject, when you call a Show you should specify what document the function must render, the url of a Show block will have the following pattern.

http://CouchDBhost:port/database/_design/designDocument/_show/showName/documentId

To carry out this task we are going to design a show for an Album and we will move our application away from internal jQuery mobile pages. Open your Smalltalk browser and start adding a method #initShows to the Albums class.

```
Albums initShows
self addShowAt: 'album' put: [:aDoc :req|
|html|
html:='<div data-role="page" id="albumview">
          <div data-role="header" id="albumheader">
               <h2 class="albumtitle">', (aDoc at:'title') , '</h2>
          </div>
          <div data-role="content" id="albuncontent">
               <h2 class="artist">', (aDoc at:'artist'), '</h2>
```

```
                    <p class="title">', (aDoc at:'title'), '</p>
                    <p class="description">', (aDoc at:'description'),'</p>
            </div>
            <div data-role="footer"/>
</div>'.
].
initialize
self appName: 'albums'.
self language: 'smalltalk'.
self initViews.
self initShows.
self importAllAttachmentsFrom:
'D:\smalltalk\SCouchViewServer\Contents\Resources\attachments\'.
self connectTo:'localhost' port:5984 userName:'admin' password:'********'.
self upload.
```

Now that we have the show function serving up our album view pages. We need to clean up our Index.html page modifying the following block inside the for..next loop.

```
…
description =  album.description;
externalPage = "_show/album/"+ album._id;
listItem ="<li class=\"album\">"       +
      "<a href=\""+externalPage+"\">"+
      "<h2 class=\"artist\">" + artist +"<\/h2>" +
      "<\/a>" +
      "<p class=\"title\">" + title +"<\/p>" +
      "<p class=\"description\">" + description +"<\/p>";
$("#albums").append(listItem);
…
```

Delete the openAlbum function and the binding to the pageBeforeShow event, delete also the second jQuery mobile page. Save them all and upload the application again

```
Workspace
      Albums new [alt+d]
```

Showing documents with Mustache

Mustache is included in the SCouchViewServer repository, Mustache is ported from the javascript source code to Smalltalk to give Smalltalk the power of templates.

Let’s start porting our simple Album show page to Mustache, copy your html string content to a file called …/resources/templates/album.mustache

```
<div data-role="page" id="albumview" data-position="inline" data-back="true">

      <div data-role="header" id="albumheader">
            <h1 class="albumtitle"> {{title}} </h1>
             <a href="#home" data-icon="grid" class="ui-btn-right">Home</a>
      </div>
      <div data-role="content" id="albuncontent" data-identity="{{document}}">
      <h2 class="artist">{{artist}}</h2>
      <p class="title">{{title}}</p>
      <p class="description">{{description}}</p>
      </div>
      <div data-role="footer"/>
</div>
```

Pay attention we have added a “home” anchor in our template and that’s the reason we have to assign the id=’home’ to the page div in our index.html file.
We need to ask to Album SCouchDBApp to load the templates from the templates folder

```
Albums initialize
self appName: 'albums'.
self language: 'smalltalk'.
self initViews.
self initShows.
self importAllAttachmentsFrom:
'D:\smalltalk\SCouchViewServer\Contents\Resources\attachments\'.
self importAllTemplatesFrom:
'D:\smalltalk\SCouchViewServer\Contents\Resources\templates\'.
self connectTo:'localhost' port:5984 userName:'admin' password:'********'.
self upload.
```

And change the album show method code to:

```
Albums initShows
self addShowAt: 'album' put: [:aDoc :req|
	|stash template|
	template:=self templates  at:'album'.
	stash:= JsonObject fromAssociations:{
		'artist' -> (aDoc at:'artist').
		'title' ->( aDoc at:'title').
		'description' -> (aDoc at:'description').
		'document' -> (aDoc at:'_id')
	}.
	Mustache toHtml:template view:stash.
	].
```

Partial documents with Mustache

Now write with your favorite editor the following file and save it on /attachments/scripts/album-page.js, the purpose of this script is avoid incongruences in our web page interface and CouchDB data, because of the jQuery cache.

```
var AlbumPageController = function(){
	function handleView(){
		//Watch for bound hide of page to clear from cache.
		var docId = $("#albumcontent").data("identity");
		var albumPage=$(document.getElementById("_show/album/" + docId));
		albumPage.bind("pagehide", handlePageViewHide);
	}

	function handlePageViewHide(){
		var docId = $("#albumcontent").data("identity");
		var albumPageCache=$(document.getElementById("_show/album/" + docId));
		albumPageCache.unbind("pagehide", handlePageViewHide);
		albumPageCache.empty();
		albumPageCache.remove();
	}

	return {
		initialize :function(){
			$("div[data-role='page']").live("pageshow",function(){
				$("div[data-role='page'"]).die("pagesshow");
				});
			}
		};
	}();

function handlePageViewReady(){
	AlbumPageController.initialize();
	}
$().ready(hadlePageViewReady);
}
```

Let’s open another new file at templates/script/scripts.html with the following content.

```
<script src="../../scripts/album-page.js"></script>
```

Let’s add the next line at the end of the album.mustache file

```
…
{{>script/script}}
```

When you import the whole templates folder content to a SCouchDBApp instance, SCouchDBApp creates a new field called “templates” and a JsonObject node per file within the root directory called “filename” without the extension, the import method goes over all the subfolders and repeats the procedure but the given name for each node will be subfolder/subfolder/…/filename and all of this nodes will be placed in a subnode called partials.

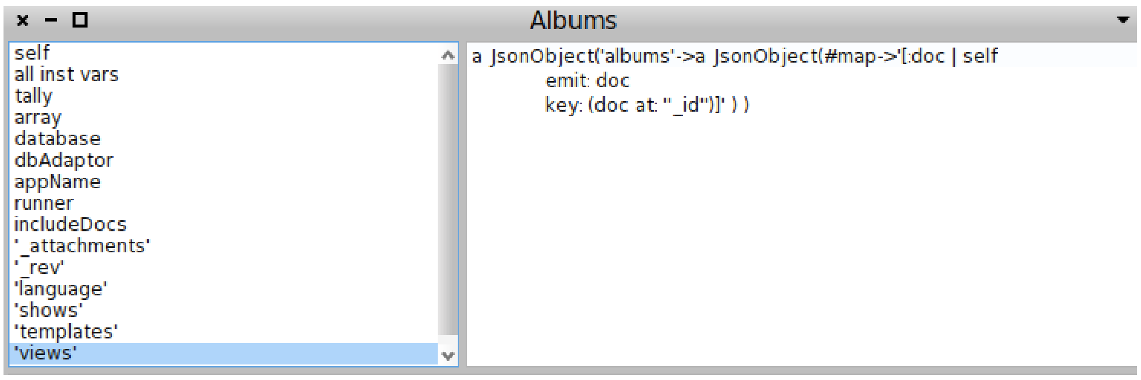
In order to include a **partial** content you have to write the node name between {{>…}}. Mustache needs the complete partials node passed as argument in our show block.

```
self addShowAt: 'album' put: [:aDoc :req|
	|stash template|
	template:=self templates at:'album'.
	stash:= JsonObject fromAssociations:{
		'artist' -> (aDoc at:'artist').
		'title' ->( aDoc at:'title').
		'description' -> (aDoc at:'description').
		'document' -> (aDoc at:'_id')
	}.
	Mustache toHtml:template view:stash partials: self partials.
	].
```

Take a breath. Inspect the application

As we’ve said before, SCouchDBApp are json documents, they have a _rev field, and the _id will be ('_design/', self appName). SCouchDBApp Object maps the Design document within the database as in CouchApp[1].

Smalltalk tools inspect:(Albums new).[Alt+d]

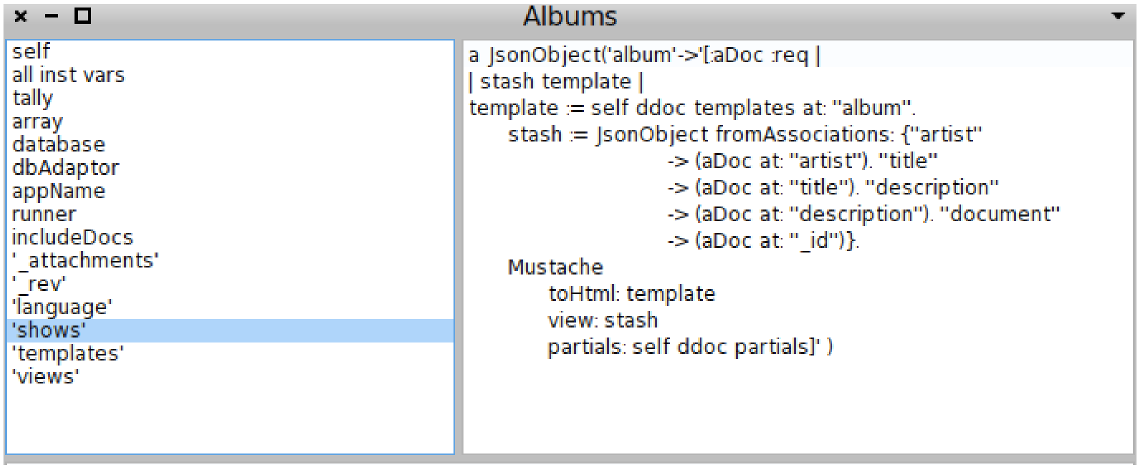


```
self
all inst vars
tally
array
database
dbAdaptor
appName
runner
includeDocs
'_attachments'
'_rev'
'language'
'shows'
'templates'
'views'
```

```
a JsonObject('albums'->a JsonObject(#map->'[:doc | self
		emit: doc
		key: (doc at: ''_id'')]' ) )
```

Current Views of our Albums applications

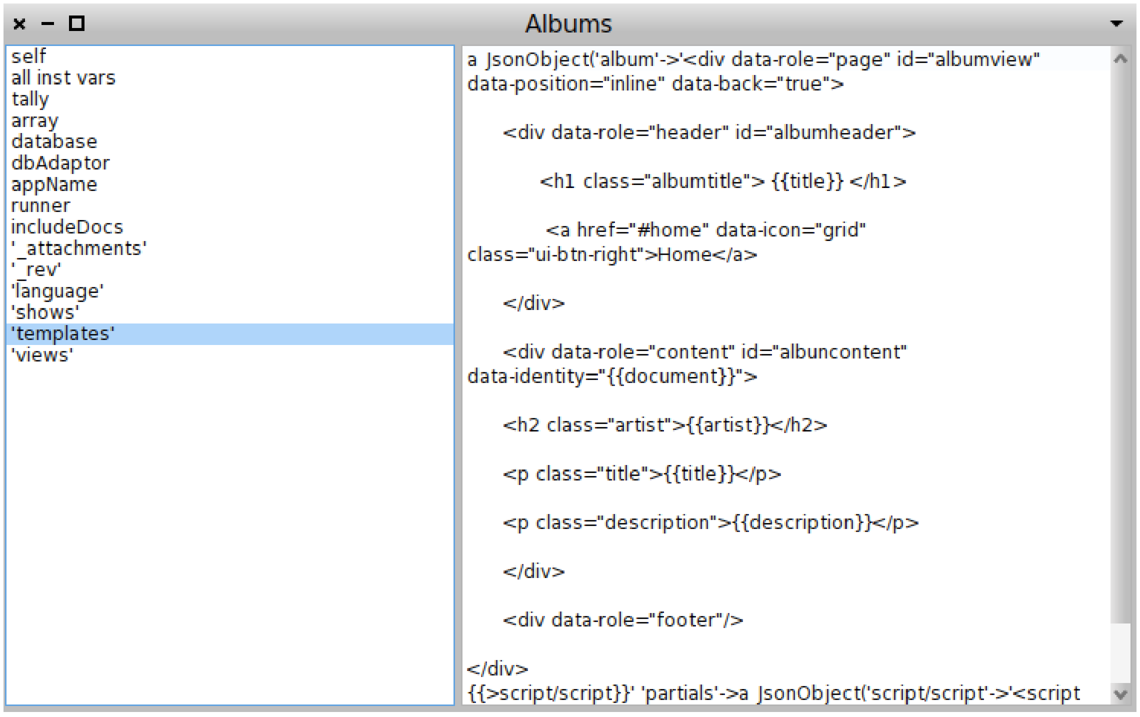
[1] CouchApp is a python utility to build couch applications



```
self
all inst vars
tally
array
database
dbAdaptor
appName
runner
includeDocs
'_attachments'
'_rev'
'language'
'shows'
'templates'
'views'
```

```
a JsonObject('album'->'[:aDoc :req |
| stash template |
template := self ddoc templates at: ''album''.
    stash := JsonObject fromAssociations: {''artist''
                    -> (aDoc at: ''artist''). ''title''
                    -> (aDoc at: ''title''). ''description''
                    -> (aDoc at: ''description''). ''document''
                    -> (aDoc at: ''_id'')}.
    Mustache
        toHtml: template
        view: stash
        partials: self ddoc partials]' )
```

Show blocks

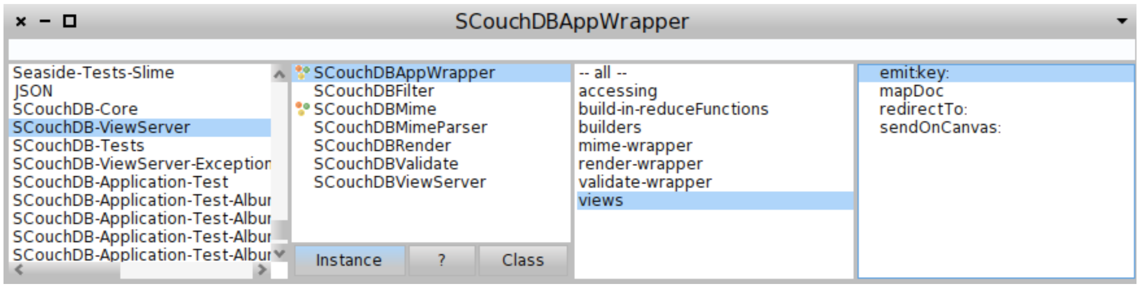


```
self
all inst vars
tally
array
database
dbAdaptor
appName
runner
includeDocs
'_attachments'
'_rev'
'language'
'shows'
'templates'
'views'
```

```
a JsonObject('album'-><div data-role="page" id="albumview"
data-position="inline" data-back="true">

    <div data-role="header" id="albumheader">

        <h1 class="albumtitle"> {{title}} </h1>

         <a href="#home" data-icon="grid"
class="ui-btn-right">Home</a>

    </div>

    <div data-role="content" id="albuncontent"
data-identity="{{document}}">

    <h2 class="artist">{{artist}}</h2>

    <p class="title">{{title}}</p>

    <p class="description">{{description}}</p>

    </div>

    <div data-role="footer"/>

</div>
{{>script/script}}' 'partials'->a JsonObject('script/script'->'<script
```

Mustache Templates and partials

All the view, List, show and update and validation blocks are compiled within a SCouchDBAppWrapper instance context, that means you can call all the methods of this class using self, you are able to call emit:key: inside a view block, getRow in a list block, redirect in a show block. As you can see, it should be easy to add new reduce methods, basically you have to create a new method that returns a BlockClosure with the reduce block and add it to the Build-in Reduce functions dictionary as you can see in the image below. You can use a build in reduce function in your application using the method SCouchApp #addViewAt:aMapFunctionName put:aMapFunction reduce:aReduceFunction



SCouchDBAppWrapper

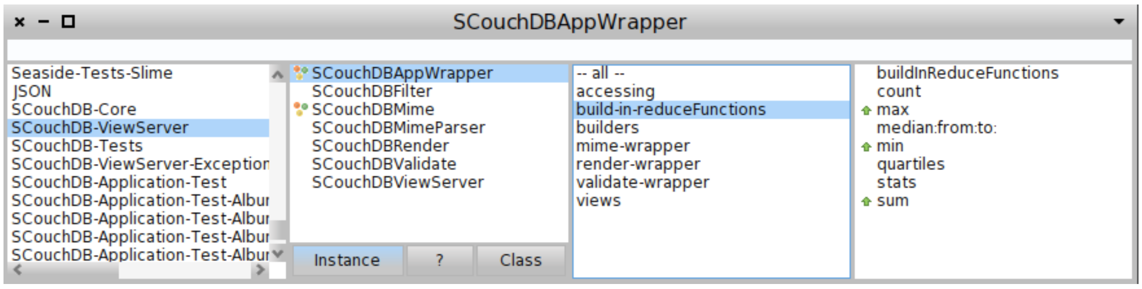
Seaside-Tests-Slime
JSON
SCouchDB-Core
SCouchDB-ViewServer
SCouchDB-Tests
SCouchDB-ViewServer-Exception
SCouchDB-Application-Test
SCouchDB-Application-Test-Albur
SCouchDB-Application-Test-Albur
SCouchDB-Application-Test-Albur
SCouchDB-Application-Test-Albur

SCouchDBAppWrapper
SCouchDBFilter
SCouchDBMime
SCouchDBMimeParser
SCouchDBRender
SCouchDBValidate
SCouchDBViewServer

-- all --
accessing
build-in-reduceFunctions
builders
mime-wrapper
render-wrapper
validate-wrapper
views

emit:key:
mapDoc
redirectTo:
sendOnCanvas:

SCouchDBAppWrapper context



SCouchDBAppWrapper

Seaside-Tests-Slime
JSON
SCouchDB-Core
SCouchDB-ViewServer
SCouchDB-Tests
SCouchDB-ViewServer-Exception
SCouchDB-Application-Test
SCouchDB-Application-Test-Albur
SCouchDB-Application-Test-Albur
SCouchDB-Application-Test-Albur
SCouchDB-Application-Test-Albur

SCouchDBAppWrapper
SCouchDBFilter
SCouchDBMime
SCouchDBMimeParser
SCouchDBRender
SCouchDBValidate
SCouchDBViewServer

-- all --
accessing
build-in-reduceFunctions
builders
mime-wrapper
render-wrapper
validate-wrapper
views

buildInReduceFunctions
count
max
median:from:to:
min
quartiles
stats
sum

Build in reduce methods

Edit/Add/delete your documents

In this section there is nothing new to add regarding the development of applications with Smalltalk and CouchDB so that I'll include only the final files. For a detailed explanation, read the jQuery mobile+CouchDB tutorial. Download the source code from http://custardbelly.com/downloads/couchapp/jqm_CouchDB_albums.zip and copy the whole template folder content, and the album-xxxx.js, this files are templates for the shows blocks and its javascript controller, we have one pair of files per edit/Add/delete roles.

We have to change the last line in each template file {{>scripts}} to {{>albumdelete/scripts}} p.e. because Mustache partials are stored in a different way in our Smalltalk Mustache implementation.

In a Javascript project, templates node has a tree structure

```
Templates
    A
    B
    Partials
        Subdir
            C
            D
...
```

On the other hand Partials in Smalltalk are much more like the attachments node

```
Templates
    A
    B
    Partials
        Subdir/C
        Subdir/D
```

The height of the Mustache tree node is always 2. When you call the Mustache #toHtml: view: partials: you always have to specify the same partial node and when you reference the partial file in a template, you have to specify the whole route to your partial file.

Let's open the Smalltalk image and port the show functions code from javascript to Smalltalk.

```
album.js
function(doc, req) {
    var mustache =
require("vendor/couchapp/lib/mustache");
    var stash = {
                artist: doc.artist,
                title : doc.title,
                description: doc.description,
                document: doc._id
  };
    return
Mustache.to_html(this.templates.album,
stash, this.templates.partials.album);
}
```

```
self addShowAt: 'album' put: [:aDoc :req|
    |stash template|
    template:=self templates at:'album'.
    stash:= JsonObject fromAssociations:{
        'artist' -> (aDoc at:'artist').
        'title' ->( aDoc at:'title').
        'description' -> (aDoc at:'description').
        'document' -> (aDoc at:'_id')
    }.
    Mustache toHtml:template view:stash partials:
self partials.
    ].
```

```
album-delete.js
function(doc, req) {
    var mustache =
require("vendor/couchapp/lib/mustache");
    var stash = {
                artist: doc.artist,
                title : doc.title,
                document: doc._id
  };
    return
Mustache.to_html(this.templates.albumdelete,
stash, this.templates.partials.albumdelete);
}
```

```
self addShowAt: 'album-delete' put:
[:aDoc :req|
    |stash template|
    template:=self templates at:'album-delete'.
    stash:= JsonObject fromAssociations:{
        'artist' -> (aDoc at:'artist').
        'title' ->( aDoc at:'title').
        'document' -> (aDoc at:'_id')
    }.
    Mustache toHtml:template view:stash partials:
self partials.
    ].
```

```
album-edit.js
function(doc, req) {
    var mustache =
require("vendor/couchapp/lib/mustache");
    var stash = {
                artist: doc.artist,
                title : doc.title,
                description: doc.description,
                document: doc._id
  };
    return
Mustache.to_html(this.templates.albumedit,
stash, this.templates.partials.albumedit);
}
```

```
self addShowAt: 'album-edit' put: [:aDoc :req|
    |stash template|
    template:=self templates at:'albumedit'.
    stash:= JsonObject fromAssociations:{
        'artist' -> (aDoc at:'artist').
        'title' ->( aDoc at:'title').
        'description' -> (aDoc at:'description').
        'document' -> (aDoc at:'_id')
    }.
    Mustache toHtml:template view:stash partials:
self partials.
    ].
```

The application folder must be like this:

```
albums
        attachments
                script
                        jquery.albums.app.js
                        jquery.albums.loginDialog.js
                        command_queue.js
                        album-page.js
                        album-delete-dialog.js
                        album-edit-page.js
                style
                        images
```

```
                    …jquery images
               jquery.mobile-1.0a2.css
               main.css
          vendor
               couchapp
                    loader.js
                    jquery.pathbinder.js
                    jquery.mobile-1.0a2.js
                    jquery-1.4.4.js
                    jquery.mustache.js
                    jquery.couch.app.util.js
                    jquery.couch.app.js
                    jquery.evently.js
                    jquery.tmpl.js
          index.html
     templates
          album
               scripts.html
          albumdelete
               scripts.html
          albumedit
               scripts.html
          album.mustache (you could rename extension to mustache or to let html)
          albumdelete.mustache
          albumedit.mustache
```

Document validation

The last thing we have to port to complete the album application in our environment is create a validateDocUpdateFunction that will check if a new/modified album has a valid document.

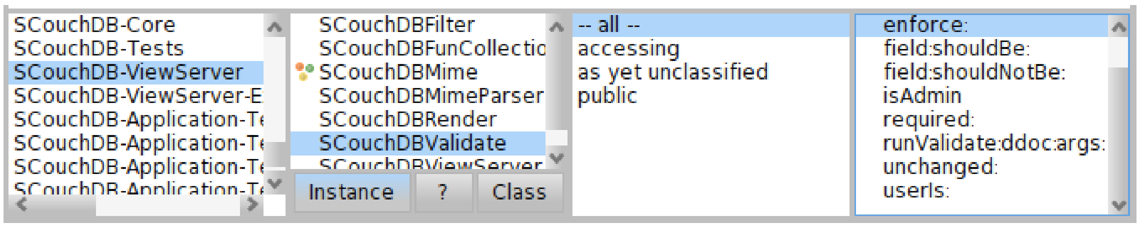
The javascript validation function included in the tutorial looks like this:

```
function( newDoc, oldDoc, userCtx ) {
  // Load validation script.
  var v = require("vendor/couchapp/lib/validate").init( newDoc, oldDoc, userCtx );

  // Create method to test if valid user.
  v.isAlbumsUser = function() {
    return v.isAdmin() || userCtx.roles.indexOf("albums-user") != -1;
  }

  // Ensure that a current session exists for editing.
  if( !userCtx.name ) {
    v.unauthorized( "You need to be logged in order to do that." );
  }
  else if( !v.isAlbumsUser() ) {
    v.forbidden( "You do not have proper access to edit this document." );
  }

  // Ensure that any updates need to match user.
  var isDeletingWithoutPermission = ( newDoc._deleted && ( oldDoc.user != userCtx.name ) );
  var isUpdatingWithoutPermission = ( newDoc.user != userCtx.name ) || ( oldDoc && ( newDoc.user
!= oldDoc.user ) );
  // If either non-permission criteria is met, checking delete first…
  if( !v.isAdmin() && isDeletingWithoutPermission ) {
    v.forbidden( "Only the creator of this document has permission to delete." );
  }
  else if( !v.isAdmin && ( !newDoc._deleted && isUpdatingWithoutPermission ) ) {
    v.forbidden( "Only the creator of this document has permission to update." );
  }
  else {
    // If it is being deleted, we are all set.
    if( newDoc._deleted ) return true;

    // Require a user field.
    v.require( "user" );
    // Ensure the assigned user is not changed.
    v.unchanged( "user" );
    // Ensure that user does not have value of undefined.
    v.assert( (newDoc.user != "undefined"), "New documents must have an associated user." );
  }
}
```

On Smalltalk will be the following BlockClosure.

```
self validateDocUpdate: [:newDoc :savedDoc :userContext|
| isDeletingWithoutPermission isUpdatingWithoutPermission isAlbumUser|
	isDeletingWithoutPermission:=false.
	isUpdatingWithoutPermission:=false.

	isAlbumUser:=(self isAdmin or:[ self userIs:'albums-user']).
	self halt.
	(userContext at:'name' )
	ifNil:[ ((UnauthorizedException new) messageText:'You need to be logged in order to do that.')
signal]
	ifNotNil:[
		(isAlbumUser) ifFalse:[((ForbiddenException new) messageText: 'You do not have proper
access to edit this document.') signal]
		].
	isDeletingWithoutPermission:=((newDoc at:'_deleted') notNil) and:[ (savedDoc user =
userContext name) not].
	isUpdatingWithoutPermission:=(newDoc user = userContext name) not or:[
		savedDoc ifNotNil:[
			(newDoc user = savedDoc user) not.
			]].

	((self isAdmin not) and:[isDeletingWithoutPermission] )
	ifTrue:[

		((ForbiddenException new ) messageText:'Only the creator of this document has permission
to delete') signal
	] ifFalse:[
		((self isAdmin  not) and:[isUpdatingWithoutPermission and:[((newDoc at:'_deleted') =
true) not ]] )
		 ifTrue:[
				((ForbiddenException new) messageText: 'Only the creator of this document has
permission to update') signal
				]
		ifFalse:[
				((newDoc at:'_deleted') = true)
				ifFalse:[	self required:'user'.
						self unchanged: 'user'.]

			]
	]
]
```

A validation block closure will be evaluated in a SCouchDBAppWrapper context that wraps the SCouchDBValidate object messages you need, it will provide a set of build-in messages to make easy the validation process, the image shows us the available methods.



enforce:
field:shouldBe:
field:shouldNotBe:
isAdmin
required:
runValidate:ddoc:args:
unchanged:
userIs:

Build-in validation methods

#enforce:aBlock	Imposes “aBlock” to be true.
#field:aField souldBe:aValue	Checks if a field value is equal to “aValue” if not it will raise a ForbiddenException.
#field:aField souldNotBe:aValue	Checks if a field value is not equal to “aValue” if true it will raise a ForbiddenException.
#isAdmin	true if the session user rol is “_admin” otherwise false.
#required:aField	Checks if a field exists and is not nil otherwise rises a ForbiddenException.
#unchanged:aField	Checks if a field has changed from the last saved document otherwise rises a ForbiddenException.
userIs:aRol	true if the user has “aRol” otherwise false.

Redesigning our Application

Now it’s the time to break away from the Todd Anderson’s tutorial to rewrite our album application. There are many reasons to rewrite the application because at the time of this writing the tutorial by Todd Anderson is not running in the last jQuery mobile version, in this tutorial, the author solves a cache problem with a set of functions too much complicated and creates “artificial” footers because he can’t manage the createPage event in a correct way. Mainly we want to drive our application with the latest jQuery mobile version and the Evenly library, externalize all of the jQuery pages and use a CouchDB Lists there where a list is required.

First of all let’s create a template file for our album list. Mustache provide us the useful {{#list}} block {{/list}} to make ease a loop into an array of documents.

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Albums</title>
  <link rel='stylesheet' href="/albums/_design/albums/vendor/couchapp/jquery.mobile-1.1.0.css"
type="text/css"/>
  <link rel="stylesheet" href="/albums/_design/albums/vendor/couchapp/jquery.mobile.structure-
1.1.0.css" type="text/css"/>
  <link rel="stylesheet" href="/albums/_design/albums/vendor/couchapp/jquery.mobile.theme-
1.1.0.css" type="text/css"/>
 </head>
 <body>
  <div id="home" data-role="page"  >
    <div data-role="header" data-position="fixed"><h1>Albums</h1></div>
    <div data-role="content" class="command-no-cache">
      <ul id="albums" data-role="listview" data-theme="c" data-dividertheme="b">
     {{>albumindex/albums}}
     </ul>
    </div>
    <div data-role="footer" data-position="fixed">
```

```
        <div data-role="navbar">
          <ul class="ui-grid-a">
            <li style="width:100%;"><a href="/albums/_design/albums/_show/album-add" data-
transition="slideup" data-icon="plus">Add Album</a></li>
          </ul>
        </div>
      </div>
    </div>

<script src="../../vendor/couchapp/loader.js"></script>
<script>

    $.couch.logout();
    $db = $.couch.db("albums");

    function handleDocumentReady()
    {
        // Set database reference and dialog template on albums.
        $.extend( $.albums, {
            database: $db,
            dialog:  $("#loginSignupDialog")
           });


      $("div[data-role*='page']").live("pagehide", function(event, ui){
        if ($(this).children("div[data-role*='content']").is(".command-no-cache"))  $(this).remove();
        });
    }

    function handlePageAlbumViewReady(){
            //Evently album page
            $.couch.app(function(app){$("#albumFooter").evently("albumFooter",app);});
            $.couch.app(function(app){$("#albumView").evently("albumView",app);});
            $.evently.connect("#albumFooter","#albumView",["albumDelete","albumEdit","albumLet"]);
    }


    function handlePageAlbumDeleteReady(){
            //Evently album delete dialog
            $.couch.app(function(app){$("#deleteFooter").evently("deleteFooter",app);});
            $.couch.app(function(app){$("#albumDelete").evently("albumDelete",app);});
            $.evently.connect("#deleteFooter","#albumDelete",["deleteCancel","deleteConfirm"]);

    }

    function handlePageAlbumLoginReady(){
            //Evently album login dialog
            $.couch.app(function(app){$("#loginCtrl").evently("loginCtrl",app);});
            $.couch.app(function(app){$("#loginDialog").evently("loginDialog",app);});
            $.evently.connect("#loginCtrl","#loginDialog",["loginCancel","loginConfirm"]);

    }

    function handlePageAlbumEditReady(){
            //Evently album edit dialog
            $.couch.app(function(app){$("#editFooter").evently("editFooter",app);});
            $.couch.app(function(app){$("#albumEdit").evently("albumEdit",app);});
            $.evently.connect("#editFooter","#albumEdit",["editCancel","editConfirm"]);

    }


    function handlePageAlbumAddReady(){
            //Evently album add dialog
            $.couch.app(function(app){$("#addFooter").evently("addFooter",app);});
            $.couch.app(function(app){$("#albumAdd").evently("albumAdd",app);});
            $.evently.connect("#addFooter","#albumAdd",["addCancel","addConfirm"]);

    }

    $("#albumView").live("pagebeforecreate", handlePageAlbumViewReady );
    $("#albumDelete").live("pagebeforecreate", handlePageAlbumDeleteReady );
     $("#loginDialog").live("pagebeforecreate", handlePageAlbumLoginReady );
    $("#albumEditPage").live("pagebeforecreate", handlePageAlbumEditReady );
    $("#albumAddPage").live("pagebeforecreate", handlePageAlbumAddReady );
    $(document).ready( handleDocumentReady );

 </script>
</html>
```

The partial file inserted in the main index template must be saved in templates/albumindex/albums.mustache

```
{{#albums}}
<li class="album">
    <a href="/albums/_design/albums/_show/album/{{_id}}">
    <h2 class="artist"> {{artist}} </h2>
     </a>
     <p class="title">{{title}}</p>
     <p class="description"> {{description}}</p>
</li>
{{/albums}}
```

The empathized function in bold will remove the page from the jQuery mobile cache, when the page div class is “command-no-cache” and solves the problem caused by the jQuery mobile cache when the page is updated. Only two pages need no-caching function, the album list after an insertion, and the Album details page after a document update.

Now let’s create a CouchDB List, this is a new concept in this writing, but I’m sure you will understand before you blink your eyes.

A List is a Smalltalk block closure that renders a list of documents in the “MIME” type you want. These documents will come from a view, and when a client (web browser) ask for the list url http://CouchDBhost:port/database/_design/designDocument/_list/listName/viewName

CouchDB sends these documents to the View Server one by one, each time the List block sends the #getRow message.

Let’s create a method call initList in our application that will add the albumIndex list block, it will supply the html code from a list of album documents.

```
initLists
self addListAt:'albumIndex' put:
[:head :req|
self provides:'html' with:[:aReq :aHead|
       |doc stash albums template|

       albums:=OrderedCollection new.

       [(doc:=self getRow ) isNil] whileFalse:[
              albums add: doc value.
              ].

       template:=self templates at:'albumindex'.
       stash :=JsonObject fromAssociations: {
              'albums'->(albums asArray).
       }.

       self send: (Mustache toHtml:template view:stash partials: (self partials)).
       ]
]
```

The message #provides: mimeType with: aBlock assigns a render block to a specific mime type, in the example is “html” the block receives two parameters, the Json request object and the head Json object.

We will create an ordered collection of documents in order to send a Json Objects array called “albums” to the Mustache template.

As you can see in the following table, the List API is very simple.

#getRow	Returns the current row or nil if there isn’t more rows
#send: aString	Sends an string to CouchDB,each time CouchDB sends a row, you can return the rendered row or nothing.
#sendOnCanvas: aBlock	You can render a list using an html “brush” from the seaside framework, the block will receive the html canvas to draw on it (Experimental)
#lastRow	True if the retrieved row is the last row.

Create the template files for Login, Showing, Adding, Editing, and deleting use-cases.

Templates/albumlogin.mustache

```
<div id="loginDialog" data-role="dialog" class="ui-dialog ui-body-a">
    <div class="ui-header ui-bar-a ui-corner-top ui-overlay-shadow">
     <h1 class="ui-title"></h1>
    </div>
    <div id="loginCtrl" data-role="content" class="ui-body-c ui-corner-bottom ui-overlay-shadow">
      <p>You need to be logged in to do that!</p>
      <form action="#" method="post">
        <label for="username">Username:</label>
        <input type="text" name="username" id="username" value=""  />
        <label for="password">Password:</label>
        <input type="password" name="password" id="password" value="" />
        <a id="loginConfirmButton" href="#" type="submit" data-role="button" data-
theme="b">Submit</a>
        <hr/>
        <a id="optionLinkButton" href="#" />
      </form>
    </div>
    <div data-role="footer" />
</div>
```

Templates/album.mustache

```
<div data-role="page" id="albumView" data-fetch="always" data-position="inline" data-add-back-
btn="true"   class="command-no-cache">
      <div data-role="header" id="albumheader">
           <h1 class="albumtitle">{{title}}</h1>
           <a href="#home" data-icon="grid" class="ui-btn-right">Home</a>
      </div>
      <div data-role="content" id="albumcontent" data-identity="{{document}}">
           <h2 class="artist">{{artist}}</h2>
           <p class="title">{{title}}</p>
           <p class="description">{{description}}</p>
      </div>
      <div  data-role="footer" data-position="fixed">
           <div id="albumFooter" data-role="navbar">
           <ul class="ui-grid-a">
                     <li><a href="#" id="deleteButton" data-icon="minus">Delete</a></li>
                     <li><a href="#" id="editButton" data-icon="gear" data-theme="b">Edit</a></li>
           </ul>
           </div>
```

```
    </div>

</div>
```

Templates/albumdelete.mustache

```
<!-- Class Styles borrowed from those applied to mobile.dialog from jquery.mobile -->
<div data-role="dialog" id="albumDelete"  data-backbtn="false" class="ui-dialog ui-body-a">
    <!-- data-backbtn and data-nobackbtn were not working in removing the back button here. -->
    <!-- As a solution, we peaked inside the jquery mobile js and found the theme and styling that is
being applied to divs with data-role=header to fake the look. -->
    <div class="ui-header ui-bar-a ui-corner-top ui-overlay-shadow">
        <h1 class="ui-title">Delete Album?</h1>
        <!-- Borrowed from jquery.mobile plugin for assiging close button on dialog. -->
        <a id="deleteCloseButton" href="#" data-icon="delete" data-iconpos="notext" style="left:
15px; top: .4em; position: absolute;">Close</a>
    </div>
    <div data-role="content" id="deleteFooter" data-identity="{{document}}" class="ui-body-c ui-
corner-bottom ui-overlay-shadow">
        <p>Are you sure you want to delete {{artist}}, {{title}}?</p>
        <a id="deleteCancelButton" href="#" data-role="button" data-theme="a">no</a>
        <a id="deleteConfirmButton" href="#" data-role="button" data-theme="c">yes</a>
    </div>
    <div data-role="footer" />
</div>
```

Templates/albumadd.mustache

```
<div id="albumAddPage" data-role="page" data-add-back-btn="true" >
    <div data-role="header"><h1>Add Album</h1></div>
    <div id="albumAdd" data-role="content">
      <form id="albumAddForm" action="#" method="get">
        <div data-role="fieldcontain">
        <label for="artist">Artist:</label>
        <input id="addArtistField" name="artist" type="text" />
        </div>
        <div data-role="fieldcontain">
        <label for="title">Title:</label>
        <input id="addTitleField" name="title" type="text" />
        </div>
        <div data-role="fieldcontain">
          <label for="description">Description:</label>
          <textarea id="addDescriptionField" name="description" cols="40" rows="8"></textarea>
        </div>
        <div  id="footer" data-role="footer" data-position="fixed">
          <div id="addFooter" data-role="navbar">
            <ul class="ui-grid-a">
                <li><a href="#" id="addCancelButton" data-icon="delete">Cancel</a></li>
                <li><a href="#" id="addConfirmButton" data-icon="gear" data-
theme="b">Save</a></li>
            </ul>
       </div>
        </div>
      </form>
    </div>
  </div>
```

Templates/albumedit.mustache

```
<div data-role="page" id="albumEditPage" data-add-back-btn="true"  data-identity="{{document}}">
    <div data-role="header" id="albumheader">
        <h1 class="albumtitle">{{title}}</h1>
    </div>
    <div id="albumEdit" data-role="content" data-theme="c">
        <!-- Filled by Evently-->
    </div>
    <div id="footer" data-role="footer" data-position="fixed" >
        <div id="editFooter" data-role="navbar">
            <ul class="ui-grid-a">
```

```
                <li><a href="#" id="editCancelButton" data-icon="delete">Cancel</a></li>
                <li><a href="#" id="editConfirmButton" data-icon="gear" data-
theme="b">Save</a></li>
            </ul>
        </div>
    </div>
</div>
```

Each template is used in the corresponding show block.

```
initShows
self addShowAt: 'album-login' put: [:aDoc :req|
	|template|

	template:=self templates at:'albumlogin'.
	Mustache toHtml:template view:aDoc partials: (self partials).
	].

self addShowAt: 'album' put: [:aDoc :req|
	|stash template|

	template:=self templates at:'album'.
	stash:= JsonObject fromAssociations:{
		'artist' -> (aDoc at:'artist').
		'title' ->( aDoc at:'title').
		'description' -> (aDoc at:'description').
		'document' -> (aDoc at:'_id')
	}.
	Mustache toHtml:template view:stash partials: (self partials).
	].

self addShowAt: 'album-delete' put:
[:aDoc :req|
	|stash template|
	template:=self templates at:'albumdelete'.
	stash:= JsonObject fromAssociations:{
		'artist' -> (aDoc at:'artist').
		'title' ->( aDoc at:'title').
		'document' -> (aDoc at:'_id')
	}.
	Mustache toHtml:template view:stash partials: (self partials).
	].


self addShowAt: 'album-edit' put: [:aDoc :req|
	|stash template|
	template:=self templates at:'albumedit'.
	stash:= JsonObject fromAssociations:{
		'artist' -> (aDoc at:'artist').
		'title' ->( aDoc at:'title').
		'description' -> (aDoc at:'description').
		'document' -> (aDoc at:'_id')
	}.
	Mustache toHtml:template view:stash partials: (self partials).
	].

self addShowAt: 'album-users' put: [:aDoc :req|
	| template adminUsers userCollection stash|

	adminUsers:=((req at:'secObj') at:'admins') at:'names'.
	userCollection:=OrderedCollection new.
	adminUsers do:[:anUserName|
		userCollection add: (
			JsonObject fromAssociations: {'userName'->anUserName})
		].


	stash:= JsonObject fromAssociations:{
		'artist' -> (aDoc at:'artist').
		'title' ->( aDoc at:'title').
		'description' -> (aDoc at:'description').
```

```
            'document' -> (aDoc at:'_id').
            'admins'->(userCollection asArray)
        }.
        template:=self templates at:'albumusers'.

        Mustache toHtml:template view:stash partials: self partials.
        ].

self addShowAt: 'album-add' put:
[:aDoc :req|
        |stash template|
        template:=self templates at:'albumadd'.
        Mustache toHtml:template view:stash partials: (self partials).
        ].
```

Evently. Browse around the application

Evently is a jQuery plugin designed to write event-based applications. SCouchViewServer supports the addition of Evently to our application design document, although importing an evently folder tree from the file system is already supported sending the message #importAllEventlyFilesFrom:aFolderPath to our SCouchDBApp instance, we are able to declare an Evently tree creating using the SCEvently class. SCEvently is a subclass of JsonObject.

An Evenly driven application declares an initial content in each div tag you need to update via evently also declares what events have to listen, and what have to show inside a div tag after the event is received. These events are emitted by a set of Dom elements defined by a jQuery selector responding on a standard jQuery event.

You can find an example on using Evently based on the http://couchapp.org/page/evently-do-it-yourself tutorial. I suggest you to follow this nice tutorial.

Let’s start defining what events emit when a button receives a click event such as:

```
(self evently at:'addFooter' onEvent:'_init')
      selector: '#addCancelButton'
      onClick:'function(){
                  $(this).trigger(''addCancel'');
                  }';
      selector: '#addConfirmButton'
      onClick:'function(){
                  $(this).trigger(''addConfirm'');
                  }'.
```

This source code declares two event listeners attached to #addCancelButton and #addConfirmButton that will trigger the events addCancel and addConfirm. Now we must declare who is listening to those events and what it has to do with them.

initEventlyAlbumAdd

```
(self evently at:'albumAdd' )
            onEvent:'addCancel' do:
            'function( event ){
                  $.mobile.changePage( "/albums/_design/albums/_list/albumIndex/albums", "slide",
true, true );
            }';
            onEvent:'addConfirm' do:
            'function(e){
            var document = {};

            ...
```

The div tag named “albumAdd” will react on receiving addCancel event, changing the jquery mobile page to our list block "/albums/_design/albums/_list/albumIndex/albums" with a slide effect.

Sometimes we are interested in what to show the first time a div tag is loaded, so we have to use the #mustache: data: message. The javascript function inside data: must return a model for the mustache template and mustache: must be a Mustache template String.

initEventlyAlbumEdit

```
(self evently at:'albumEdit' )
        …
        onEvent:'_init'
          data:'function(e){
               var docId = $("#albumEditPage").data("identity");
               var currentAlbum;
               $db.openDoc( docId, {
                         success: function( document ) {
                              currentAlbum=document;

                         },
                         error: function() {
                              alert( "Cannot open document: " + docId );
                         }
                    },
                    {async:false});
               $$(this).currentAlbum = currentAlbum;
               return  $$(this).currentAlbum;
          }'
          mustache:
          '<form id="albumform" action="#" method="get" >
               <div data-role="fieldcontain">
               <label for="artist">Artist:</label>
               <input id="artistField" name="artist" type="text" value="{{artist}}" />
               </div>
               <div data-role="fieldcontain">
               <label for="title">Title:</label>
               <input id="titleField" name="title" type="text" value="{{title}}" />
               </div>
               <div data-role="fieldcontain">
                    <label for="description">Description:</label>
                    <textarea id="descriptionField" name="description" cols="40"
rows="8">{{description}}</textarea>
               </div>
          </form>'
```

We still have to add some javascript lines to our page to configure Evently. Basically we have to add the lines below.

```
function handleEventlyPageReady(){
        //Evently page
        $.couch.app(function(app){$("#aa").evently("aa",app);});
        $.couch.app(function(app){$("#bb").evently("bb",app);});
        $.evently.connect("#aa","#bb",["event1","event2"]);
}
$("#page_id").live("pagebeforecreate", handleEventlyPageReady );
```

The main class in the Evently package is SCEvently, it has a set of methods that will help us to build an Evently tree and specify the reactions to events. We can find two classes more in the Evently package, SCEventlyQuery and SCEventlySelector, this classes will be instantiated by SCEvently with the method #query and #selector: ,SCEventlyQuery is user to specify a CouchDB view readable inside a #data: function. You can find an example in the evently Account implementation.

```
renderItemsEventOn:evently
((evently at:'items' onEvent:'_changes')
data:'function(data) {
 // $.log(data)
 var p;
 return {
   items : data.rows.map(function(r) {
     p = (r.value && r.value.profile) || {};
     p.message = r.value && r.value.message;
     return p;
   })
 }
};'
mustache:'<p>Customize this format here:
<tt>ddoc.evently.items._changes.mustache</tt></p>
<h3>Recent Messages</h3>
<ul>
 {{#items}}
   <li>
     <div class="avatar">
       {{#gravatar_url}}<img src="{{gravatar_url}}" alt="{{name}}"/>{{/gravatar_url}}
       <div class="name">
         {{nickname}}
       </div>
     </div>
     <p>{{message}}</p>
     <div style="clear:left;"></div>
   </li>
 {{/items}}
</ul>
<p><em>Protip:</em> If you setup continuous replication between this database and a remote one,
this list will reflect remote changes in near real-time.</p>
<p>This would be a good place to add pagination.</p>')

query view:'recent-items'; descending: true;limit: 50.
```

Accessing	Browser events	Document events	Form events
at:	selector:onError:	selector:onLoad:	selector:onBlur:
at:onEvent:	selector:onResize:	selector:onReady:	selector:onChange:
at:onEvent:data:	selector:onScroll:	selector:onUnload:	selector:onFocus:
at:onEvent:data:mustache:			selector:onSelect:
at:onEvent:do:			selector:onSubmit:
at:onEvent:mustache:			
at:onEvent:mustache:data:			
at:onEvent:selector:			
data:	Keyboard events	Mouse events	
data:mustache:			
mustache:			
mustache:data:			
onEvent:	selector:onFocusIn:	selector:onClick:	
onEvent:data:	selector:onFocusOut:	selector:onDblClick:	
onEvent:data:mustache:	selector:onKeyDown:	selector:onHover:	
onEvent:do:	selector:onKeyPress:	selector:onMouseDown:	
onEvent:mustache:	selector:onKeyUp:	selector:onMouseEnter:	
onEvent:mustache:data:		selector:onMouseLeave:	
onEvent:selector:		selector:onMouseMove:	
query		selector:onMouseOut:	
selector:		selector:onMouseOver:	
selector:onEvent:do:		selector:onMouseUp:	

SCEvently methods

The accessing package is plenty of helper methods to avoid an excess of parenthesis when you specify evently nodes.

SCEvenly

#at: anEventlyName	Creates a first level node in an Evently structure and returns the node
#onEvent: anEventlyName	The same than #at: used when you specify a second level in the Evently structure with #at: firstLevel onEvent:secondLevel. Returns the second node
#data:aJScriptFunction	Adds a data node to an evently object.
#mustache:aMustacheTemplate	Adds a mustache node to an evently object.
#query	Adds a query node to an evently object.
#selector:aJQuerySelector onXXX:aJScriptFunction	Adds a selector node to an evenly object. Where XXX is a jQuery event

Updating your documents

Update blocks are useful when you need to update just a few fields of a document and are very closed to a Show but you can update the received document, in our Albums example we will add an update block because we want to register when a user gives an album to another user. Each album has a loans history.

The update handler will be the next one.

```
self addUpdateHandlerAt: 'setNewUser' put:[:aDoc :req|
	|template stash loans|

	loans:=aDoc at:'loans'.
	loans ifNil:[loans:=OrderedCollection new]
		ifNotNil:[loans:=OrderedCollection newFrom:loans].
	loans add:(JsonObject fromAssociations:{
						'user' -> ( req query newuser).
						'date' -> (DateAndTime now asString).
						}).
	aDoc at:'loans'  put:loans.
	template:=self templates at:'album'.
	stash:= JsonObject fromAssociations:{
		'artist' -> (aDoc at:'artist').
		'title' ->( aDoc at:'title').
		'description' -> (aDoc at:'description').
		'user' -> (aDoc at:'user').
		'document' -> (aDoc at:'_id').
	}.

	Array with:aDoc with:(Mustache toHtml:template view:stash partials: (self partials))
	].
```

...

An update handler block receives two arguments a document and the request object, the block must return an Array with two objects, the modified document and an optional "Show" object to forward to a client.

The way to call an update handler is asking to the server for the following url pattern:

http://CouchDBhost:port/database/_design/designDocument/_update/updateName/documentId?paramName=value

Filtering changes

The last CouchDB "trick" is the changes notification API, Filters allow you to receive only the changes that you want. To add a new filter in your design document, you have to use SCouchApp>#addFilterAt: message.

The next BlockClosure filters the changes done by the current user.

```
Albums>>initFilters
self addFilterAt:'mine'
	put:
	[:doc :req|
	self halt.
		((doc user)= (req userCtx name))
	].
```

Retrieving changes require asking to the server for the Url described below.

http://CouchDBhost:port/database/_changes?filter=designdocname/filtername

Images index