Mogreet SDK
Tutorials in Perl

proposed by Anais Brossas
October 04, 2011

Abstract

This document provides several tutorials explaining
how to install, use and modify the SKD.

Contents

1 . Introduction

The Mercury Software Development Kit (SDK) provides the functions to implement the Mogreet API Mogreet Messaging System (MoMS).

This documentation will help you to understand the SDK (developed in Perl) via tutorials and schemas explaining the global operation of the application and how to install and use it in your computer.

2 . Prerequisites

Before trying to test the SDK on a terminal, you need to check the version of perl installed on your computer.

You will need at least **Perl 5.8.**
If you don't know what version you have, open a terminal and type the following command: *perl -v*

(To install perl or to update a perl version go to www.perl.org)

Also, you will need to install these modules:

- LWP::Simple
- XML::Simple
- URI::URL

To install a module (e.g LWP::Simple), open a terminal and type:
sudo perl -MCPAN -e 'install LWP::Simple'

(If that doesn't work, go to cpan.org, search the module you need, download it and follow the instructions).

We can now launch the application.

3 . Tutorials

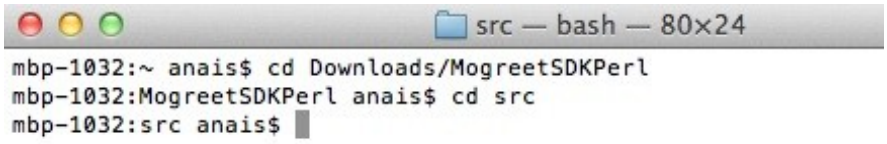
3 . 1 . How to start the application?

After decompressing the downloaded repertory, right click on it, go to Get Info. You will find its path, which is the exact location of the repertory on your computer.

Then, open a terminal, and type :
cd path/nameOfTheRepertoy

In my case, path is the directory Downloads and the name of the repertory MogreetSDKPerl.

You are now in the application directory. To execute the application, you need to go to src repertory, again type : *cd src*



The Main.pl file is the one that launch the application, to execute it, type:
perl Main.pl

You have now launched the application, you can use it directly, or follow the next tutorial to know how it works.

3 . 2 . Create a Mercury Object and Ping

After typing *perl Main.pl* on a terminal, the application want you to enter a Client Id and a Token in order to create a Mercury object. This object will allow you to execute any request to the Moms API. You will keep the same Client id and the Token during the application. If you want to change them, quit the application and relaunch it.

```
mbp-1032:src anais$ perl Main.pl
-------------------------------------------
-------------- Mogreet SDK ----------------
-------------------------------------------
--------- INITIALISE YOUR OBJECT -----------
-------------------------------------------
-------------------------------------------
------------- Enter Client Id -------------
-------------------------------------------
536
-------------------------------------------
--------------- Enter Token ---------------
-------------------------------------------
102ed2ad568f913a31aeace02eeae234
```

Now your Mercury is created, check if you entered the right Client id and the Token by doing a Ping request.

```
-------------------------------------------
-------------- Mogreet SDK ----------------
-------------------------------------------
--------- PING :          Enter 1 -----------
--------- LOOKUP :        Enter 2 -----------
--------- SEND :          Enter 3 -----------
--------- GETOPT :        Enter 4 -----------
--------- SETOPT :        Enter 5 -----------
--------- UNCACHE :       Enter 6 -----------
--------- INFO :          Enter 7 -----------
--------- TRANSACTIONS :  Enter 8 -----------
-------------------------------------------
--------- Enter any letter to quit ---------
-------------------------------------------
1

-------------------------------------------
------------------ PING -------------------
-------------------------------------------

Response:
status: success
message: pong
code: 1
```

Succes! Your ping worked, and the message sent back is pong. Every time a request works, the code is 1.
If the Ping request doesn't work, you will see an error message, and the application will quit by itself. It means that the Client id or the Token entered aren't good.

Let's try another request now! Except for the Ping, any request need at least one other information that the Client id and the token.

3 . 3 . Do a Send request

Precedent tutorials must be followed before doing this one.
You have now created a Mercury object and check its parameters by doing a Ping request. Let's now send a message via MoMS API.

Choose SEND on the menu by typing 3.
Then enter the parameters needed:

```
-----------------------------------------
-------------- Mogreet SDK --------------
-----------------------------------------
--------- PING :          Enter 1 -----------
--------- LOOKUP :        Enter 2 -----------
--------- SEND :          Enter 3 -----------
--------- GETOPT :        Enter 4 -----------
--------- SETOPT :        Enter 5 -----------
--------- UNCACHE :       Enter 6 -----------
--------- INFO :          Enter 7 -----------
--------- TRANSACTIONS :  Enter 8 -----------
-----------------------------------------
--------- Enter any letter to quit ---------
-----------------------------------------
3

-----------------------------------------
------------------ SEND -----------------
-----------------------------------------
------------ Enter Campaign Id ------------
-----------------------------------------
10651
-----------------------------------------
---------------- Enter To -----------------
-----------------------------------------
7752171448
-----------------------------------------
--------------- Enter Message -------------
-----------------------------------------
hello world
-----------------------------------------
------------- Enter Content Id ------------
-----------------------------------------
4321

Response:
status: success
message: API Request Accepted
code: 1
message Id: 38028691
Hash: rz90trlj
```

Enter you mobile phone number to receive the message

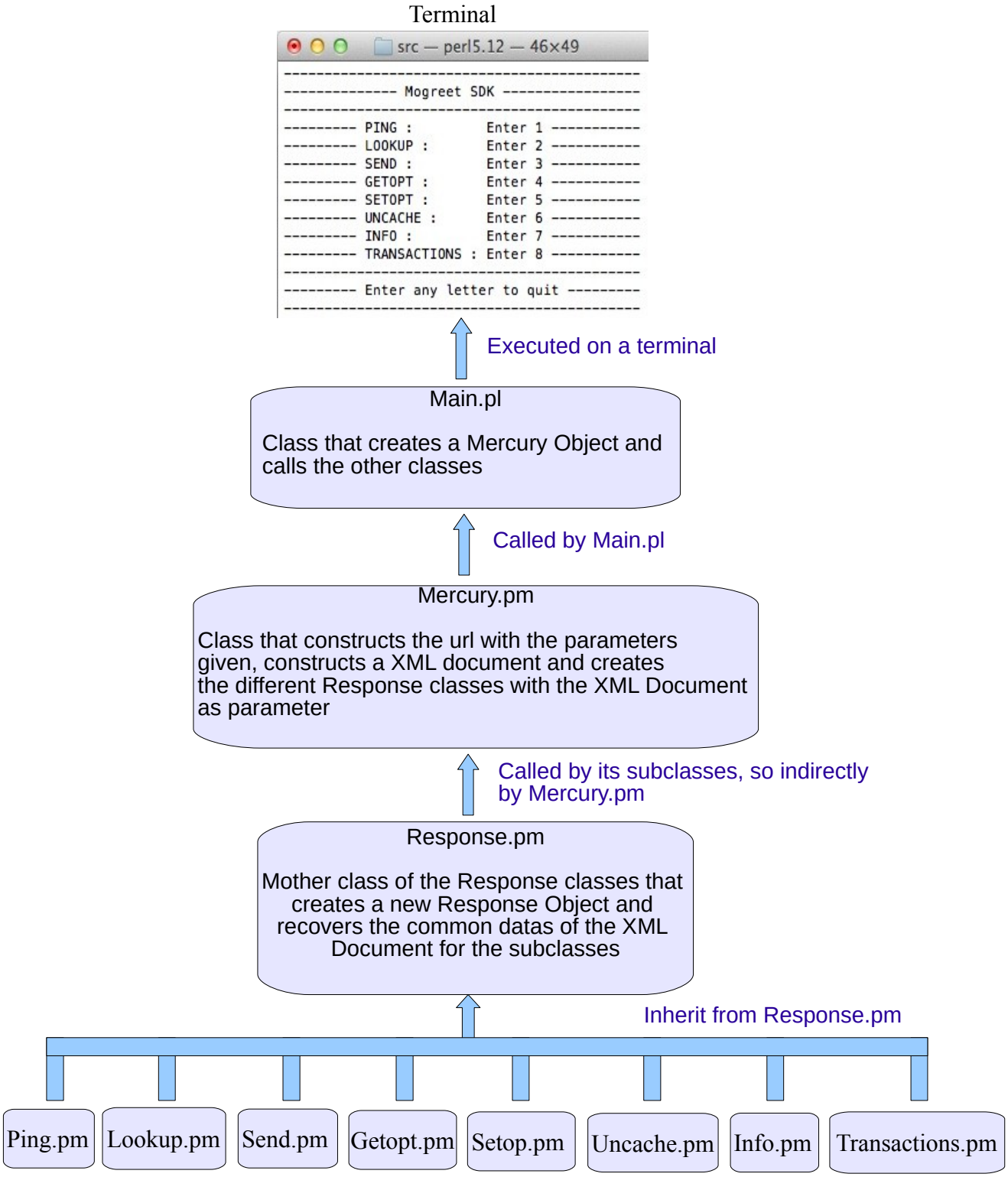
Enter any message you want

Success! The message has been sent! It is followed by a link to a video.

You know now everything about how to use the application, let's see how it works, and how you can modify it !

4 . Mogreet SDK architecture

4 . 1 . Global operation



Every subclasses calls the Response Class (which creates a new response object), and recovers the data not recovered by Response.pm

4 . 2 . Classes

4 . 2 . 1 . Main.pl

If you take a look at Main.pl, you will see that it is composed of two functions *initialiseMercury()* and *menu()*, and of the program (big *while* and *if*).
Dividing this class in three parts will allow us to understand it better.

```
1  #--- Main.pl ---
2  #!/usr/bin/perl -w
3
4  use strict;
5  use Mercury qw(:DEFAULT);
6
7  my $clientId;
8  my $token;
9  my $entre;
10 my $m;
11
12 # Creates a new Mercury Object
13 sub intialiseMercury{
14     print "-----------------------------------------\n";
15     print "-------------- Mogreet SDK ----------------\n";
16     print "-----------------------------------------\n";
17     print "--------- INITIALISE YOUR OBJECT ----------\n";
18     print "-----------------------------------------\n";
19     print "-----------------------------------------\n";
20     print "------------ Enter Client Id -------------\n";
21     print "-----------------------------------------\n";
22     $clientId = <STDIN>;
23     print "-----------------------------------------\n";
24     print "--------------- Enter Token ---------------\n";
25     print "-----------------------------------------\n";
26     $token = <STDIN>;
27
28     #checks if the client id and the token are empty or not
29     chomp($clientId);
30     chomp($token);
31     if ((length ($clientId) == 0) || (length ($token) == 0) ){
32         throw Error::Simple("Error: Client Id and Token are required!");
33     }
34
35     #creates a new Mercury Object
36     $m = new Mercury($clientId,$token);
37
38 }
39
```

Import Mercury class with its functions

Declaration of varables

Function initialiseMercury()

Recover Client Id and Token

Creates Mercury Object with Client Id and Token entered

If you don't wan to enter every time you launch the application the Client id and the token, just erase everything in the function and keep the last line (line 36) only, and enter directly the value you want.

e.g. : *$m = new Mercury('536','102ed2ad568f913a31aeace02eeae234');*

```
40 #prints list of possible requests to the MoMS API
41 sub menu {
42     print "-----------------------------------------\n";
43     print "-------------- Mogreet SDK ----------------\n";
44     print "-----------------------------------------\n";
45     print "--------- PING :         Enter 1 ----------\n";
46     print "--------- LOOKUP :       Enter 2 ----------\n";
47     print "--------- SEND :         Enter 3 ----------\n";
48     print "--------- GETOPT :       Enter 4 ----------\n";
49     print "--------- SETOPT :       Enter 5 ----------\n";
50     print "--------- UNCACHE :      Enter 6 ----------\n";
51     print "--------- INFO :         Enter 7 ----------\n";
52     print "--------- TRANSACTIONS : Enter 8 ----------\n";
53     print "-----------------------------------------\n";
54     print "--------- Enter any letter to quit --------\n";
55     print "-----------------------------------------\n";
56 }
```

This function just prints the menu

```
57
58  # calls the function that initialises the object with a token and a Client
        ID
59  intialiseMercury();
60
61  # calls the function that prints the menu with alle the kind of requests
        that can be done
62  menu();
63
64  # recovers the value entered
65  $entre = <STDIN>;
66
67  # if the value doesn't fit with the code of a request, it quits the
        application
68  if ($entre !~m/[1-8]/){
69          print "----------------- GOODBYE -----------------\n";
70  }else{ # do the request
71      while ($entre =~m/[1-8]/){
72          if ($entre == 1){ #ping request
73              print "\n-----------------------------------------\n";
74              print "------------------ PING ------------------\n";
75              print "-----------------------------------------\n";
76
77              # Ping on the object created at the beginning
78              $m->ping();
79
80          } else {
81              if ($entre == 2){ #lookup request
82                  print "\n-----------------------------------------\n";
83                  print "------------------ LOOKUP ------------------\n";
84                  print "-----------------------------------------\n";
85                  print "------------ Enter Message Id -------------\n";
86                  print "-----------------------------------------\n";
87                  my $messageId = <STDIN>;
88                  print "-----------------------------------------\n";
89                  print "--------------- Enter Hash ----------------\n";
90                  print "-----------------------------------------\n";
91                  my $hash = <STDIN>;
92
93                  # creates hash with values entered
94                  my %hash=("message_id"=>$messageId,"hash"=>$hash);
95                  # Lookup on the object created at the beginning
96                  my $lookup=$m->lookup(\%hash);
97                  print "campID---------------".$lookup->getCampaignId()."\n";
98
99              }
100             else{
101                 if ($entre == 3){ #send request
102                     print "\n-----------------------------------------\n"
                            ;
103                     print "------------------ SEND -------------------\n";
```

If you have entered 1 after the menu, it is because you want to do a ping request.
That is what is done here.
The mercury object $m you have created when launching the application, calls its ping function and creates a Ping Object.

You will see
Response :
status: success
message: pong
code: 1
This printout is handled by the Response Class.

The Ping class doesn't print anything because the XML data for a ping request are just the common data to any classes.

The Lookup class prints other datas like the campaign id, the content id, the transactions...
This prints is just due to the function *PrintElts()* in Lookup that we will see later. But if you don't want to print every data received you can:
Modify the PrintElts() function directly or delete the call to this function in the Lookup class and call the data you want directly on Main.pl.

e.g. Here (line 97) I called getCampaignId(), which returns the campaign id of the lookup request.

```
campID---------------10651
```

4 . 2 . 2 . Mercury.pm

Mercury class is composed of a constructor and 10 functions:

- *processRequest($url,$params,$reqName)* which constructs the url with the parameters given, and recovers the data by creating a XML document. Finally, it returns the document.
- *setParams()* which constructs the parameters in order to put them at the end of the url without doing any modifications in processRequest function.
- the other functions create the different response requests.

Let's take *send()* as an example to see how it works:

```
247 sub send {
248     my ($this,$refHash)=@_;
249     my $message;
250     my $to;
251     my $contentId;
252     my $campaignId;
253     my $i=0;
254
255     #constructs hash given in parameters
256     while(my($key,$value) = each %$refHash){
257         $hash{$key}=$value;
258         if ($i==0){
259             $campaignId=$value;
260         }else{
261             if($i==1){
262                 $to=$value;
263             }else{
264                 if($i==2){
265                     $message=$value;
266                 }else{
267                     $contentId=$value;
268                 }
269             }
270         }
271         $i++;
272     }
273
274     # Check if all the params contain a value
275     if (!(defined($campaignId)) || !(defined($to)) || !(defined($contentId))
        || !(defined($message))){
276         throw Error::Simple("Error: input parameter(s) missing in the SEND
            call.");
277     }
278     my $params = setParams();
279     my $xmlDoc = processRequest($SEND_URL, $params, "SEND");
280     return new Send($xmlDoc);
281 }
```

Recovers the hash with the parameters given when calling this function in Main.pl

Reconstructs the hash given in parameters because it is not possible to use it as a parameter in an other function.

This hash is saved as a class parameter, so accessible from any function (because declared at the beginning), and used in setParams() function.

Saves the Xml document created by processRequest in $xmlDocument

Creates a new Send Object (Response object) with $xmlDoc as parameter

This function, like every other is constructed following this pattern:

- Recovering of the hash
- Declaration of variables
- Hash construction
- Checking that the parameters aren't empty
- setParams() call
- processRequest() call
- Creation of a Response Object

4 . 2 . 3 . Response.pm

The Response class is composed of a constructor, getters and 3 functions:

- *printElts():* that prints the common elements of the subclasses
- *setResponseCodeStatusMessage()*

```
98  sub setResponseCodeStatusMessage {
99      try {
100         #create parser
101         $xml = new XML::Simple;
102
103         #read XML file
104         $data = $xml->XMLin($xmlDoc, forcearray => [ qw(campaign) ], keyattr=>[]);
105
106         #Set the attributes
107         $responseStatus=$data->{status};
108         $message=$data->{message};
109         $responseCode=$data->{code};
110
111         #print elements
112         printElts();
113
114     } catch Error with {
115         my $ex = shift;
116         print "\nAn error occured while getting the response code, status and message:".$ex;
117     }
118
119 }
```

If you don't want that it prints the common elements of the subclasses, delete this two lines

- *responseIsValid()* is used by the subclasses to know if the response is a success or not before continuing to execute the code.

4 . 2 . 4 . Info.pm

All the subclasses have the same architecture:

- Package (followed by the name of the class)
- Classes imported
- *our @ISA = qw(Response);* means that the class inherit of the Response Class
- Variables declarations
- Constructor
- Getters
- PrintElts() function

Here is the constructor of the Info class as an example:

```
63  sub new {
64      my ($class,$xmlDoc) = @_;
65      $class = ref($class) || $class;
66      my $this = $class->SUPER::new($xmlDoc);
67      throw Error::Simple($this->SUPER::getMessage())
68          if (!($this->SUPER::responseIsValid()));
69      try {
70          #Set the attributes
71          $number=($this->SUPER::getData())->{number};
72          $carrierId=($this->SUPER::getData())->{carrier}{id};
73          $carrier=($this->SUPER::getData())->{carrier}{content};
74          $handsetId=($this->SUPER::getData())->{handset}{id};
75          $handset=($this->SUPER::getData())->{handset}{content};
76
77          #print elements
78          printElts();
79
80          return bless($this,$class);
81
82      } catch Error with{
83          my $ex = shift;
84          print "\nAn error occured while parsing the XML data for the INFO call:".$ex;
85      }
86  }
```

Creates a new Response Object and check if responseIsValid() returns 1 (success of the request)

Sets datas in the variables created at the top of the class

Here again, you can delete the call of the printElts() function if you don't want all the elements printout on the terminal.

5 . Conclusion

I hope that this documentation helped you to install the Mogreet SDK application in Perl, understand it better, gave you few notions in Perl, allowed you to make modifications about what you wanted.

This SDK has been developed in Perl, using classes, which makes it easier to modify and to use.