Message Demo

Client Command Line

Greg Rogers

September 6, 2013

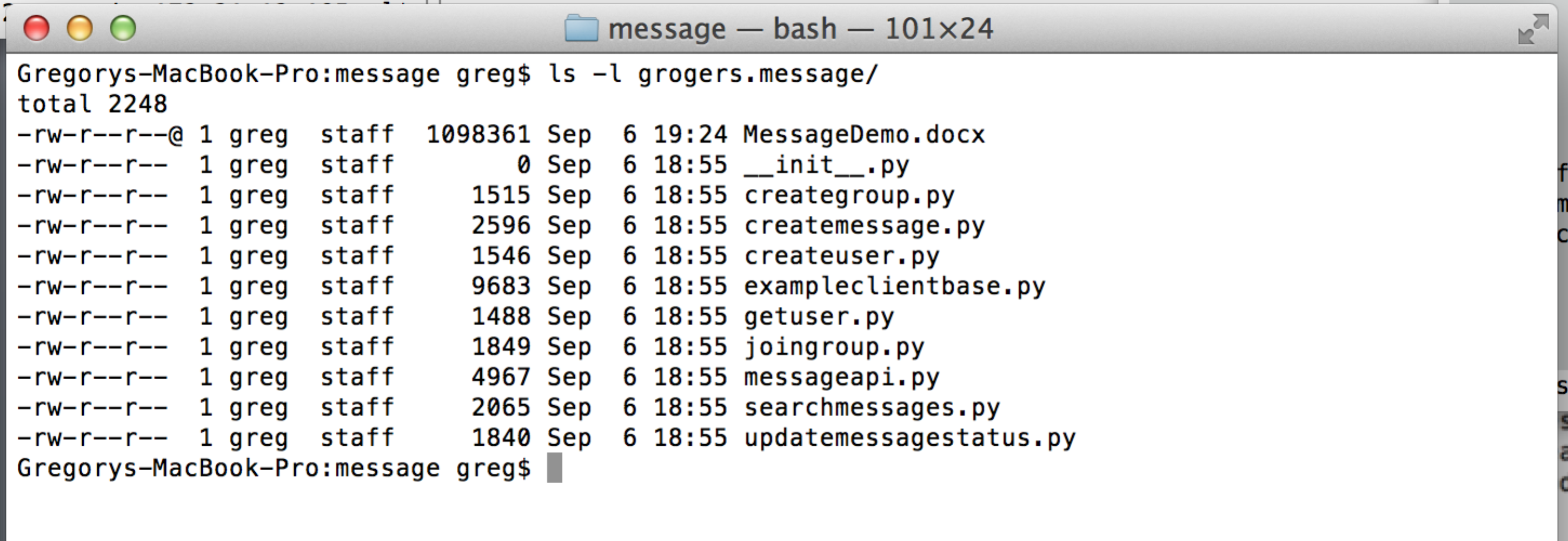
This document introduces the command line client interface to the message server demo. The examples show how the server supports user and group creation, users joining groups, messing sending between users and groups, and the progression of messages through the normal life-cycle.

Prerequisites

To run the client demo you need the following installed on your computer:

1. python 2
   1. download from <http://www.python.org/> and follow install instructions.
2. httplib2 for python
   1. <https://code.google.com/p/httplib2/wiki/Install>
   2. untar and then use “python setup.py install”
3. if you have a python version prior to 2.7 (use: “python –version” to findout)
   1. [http://argparse.googlecode.com/files/argparse-1.2.1.tar.gz](http://argparse.googlecode.com/files/argparse-1.2.1.tar.gz" \t "_blank)
   2. untar and then use “python setup.py install”
4. The message demo client scripts.
   1. http://ec2-54-200-9-5.us-west-2.compute.amazonaws.com:8080/message/static/grogers.message.tar
   2. After downloading that file, untar it with “tar xvf grogers.message.tar” The directory created will have python source files that make interaction with the message server easy.

The message demo command line files:



\_\_init\_\_.py – this file is required for python processing.

creategroup.py – use this to create a new group; you provide a name.

createmessage.py – use this to create a new message; you provide the content, user name and password for the sender, user name for a recipient and/or a group name and a message status. The default status is “draft”. You can use “send” to get the message to be sent immediately after creation.

createuser.py – use this to create a new user; you provide a name and password.

exampleclientbase.py – the base class for the other python classes. This provides the basic HTTP services. You don’t need to use this file directly.

getuser.py – retrieves the current definition of a user; with this you can see the groups a user has joined. You provide the user’s unique id or login name.

joingroup.py – use this to have a user become a member of a group. You provide the user’s name and the group name.

messageapi.py – this class provides the simple message demo API that is used by the other scripts. You don’t need to use this file directly.

MessageDemo.docx – this file.

searchmessages.py – retrieves messages from the server. The search has filter properties that all you to fetch messages by receiver and status for example. Using status “unread” gives you the “new inbox” view into a specific user’s messages. See the examples below for more details including how to page through using offset and limit inputs.

updatemessagestatus.py – changes the status of a specific message. You provide the unique id of the message and the new status. Status values are: draft, send, sending, sendingFailed, sent, unread, read, deleted. Typically a client app would change the status of just read messages from “unread” to “read”. Status of “send” signals the backend to send a copy of the message to the receiver and group members. While the message is being sent, the original message status is “sending”; when complete it is set to “sent”. These status transitions are controlled by the server. Using searchmessage.py to retrieve “unread” messages does not change their status to “read”.

Caveats on the current implementation:

When I first started this work I was thinking of group messaging more along the lines of a social network interface. Thus “searchmessages” returns a list of messages and their full content. As the implementation progressed it started to feel a bit more like email. The desired user interaction style needs to be decided and if email is the style then subject lines and other standard email properties need to be added. Likewise searchmessages should be refactored to return message subject lines and not the full content.

The following features are not yet implemented (Sept. 6th):

1. Security – the user name and password are not used for authentication. This is simple matter of adding the user and password to a basic auth header and adding the request filter definition to the server’s web.xml. The filter will pull the auth header, validate the password and refuse the request if they don’t match. On user creation the password will be encrypted using bcrypt before stored in the database. Once the request session has a validated user, several of the other requests can validate that the user has rights to the data they are requesting (e.g. allowed to manage messages they received.)
2. Receiver and Group auto-complete not yet implemented. The server will provide a REST end point to return list of user or groups with the prefix.
   1. This would use a MongoDB query like:
      1. db.users.find({loginName:/xyz.\*/},{loginName:1}).sort({loginName:1}).limit(20)
   2. where “xyz” is the prefix sent to the REST endpoint and 20 is the maximum number of matches to return.
   3. Client would use this information to fill in receiver or group reference in the message.
3. Although not visible to the client api directly. Currently message sending is done synchronously with client request to change a message status to “send”. If group membership is large, the client request might not return for a while. Clearly this doesn’t work for a production system. Sending should be handled asynchronously.
4. Error handling on the server. All responses are supposed to be JSON. A filter needs to be added request pipeline to ensure that all errors added to the response content as JSON rather than HTML.
5. Message search paging needs to be based on message dates in order to be stable as new messages are added. For example: messages created prior to a specific time and then offset and limited.

# Examples:

## Simple messaging between two users.

Gregorys-MacBook-Pro:python greg$ **python createuser.py -l greg -p secret -v**

########## request: http://ec2-54-200-9-5.us-west-2.compute.amazonaws.com:8080/message/rest/user

{u'loginName': u'greg', u'createdAt': 1378512068081, u'id': u'522a6cc4e4b0a7a09b1ff097', u'groupRefs': []}

Gregorys-MacBook-Pro:python greg$ **python createuser.py -l mimi -p secret -v**

########## request: http://ec2-54-200-9-5.us-west-2.compute.amazonaws.com:8080/message/rest/user

{u'loginName': u'mimi', u'createdAt': 1378512086707, u'id': u'522a6cd6e4b0a7a09b1ff098', u'groupRefs': []}

Two users have been created. The output shows the request sent and the resulting JSON.

Gregorys-MacBook-Pro:python greg$ **python createmessage.py -l greg -p secret -r mimi -c "hello world" -s send -v**

########## request: http://ec2-54-200-9-5.us-west-2.compute.amazonaws.com:8080/message/rest/user/name/greg

########## request: http://ec2-54-200-9-5.us-west-2.compute.amazonaws.com:8080/message/rest/user/name/mimi

########## request: http://ec2-54-200-9-5.us-west-2.compute.amazonaws.com:8080/message/rest/message/

{u'status': u'sent', u'sender': {u'id': u'522a6cc4e4b0a7a09b1ff097', u'name': u'greg'}, u'content': u'hello world', u'receiver': {u'id': u'522a6cd6e4b0a7a09b1ff098', u'name': u'mimi'}, u'id': u'522a6d0ae4b0a7a09b1ff099', u'createdAt': 1378512138372}

Here we’ve created a message and immediately sent it from greg to mimi by using the status “send”. The first two requests are GET requests to verify that “greg” and “mimi” are users and to get their user ids. You can see in the resulting JSON the which was the sender and which was the receiver and that the current status of this message greg created is “sent”.

Gregorys-MacBook-Pro:python greg$ **python searchmessages.py -l mimi -p secret -v -s unread**

########## request: http://ec2-54-200-9-5.us-west-2.compute.amazonaws.com:8080/message/rest/user/name/mimi

########## request: http://ec2-54-200-9-5.us-west-2.compute.amazonaws.com:8080/message/rest/message/search?status=unread&receiverId=522a6cd6e4b0a7a09b1ff098&limit=20&offset=0

>>>>>>>>>> search results

{

"status": "unread",

"messages": [

{

"status": "unread",

"sender": {

"id": "522a6cc4e4b0a7a09b1ff097",

"name": "greg"

},

"content": "hello world",

"receiver": {

"id": "522a6cd6e4b0a7a09b1ff098",

"name": "mimi"

},

"id": "522a6d0ae4b0a7a09b1ff09a",

"createdAt": 1378512138372

}

],

"offset": 0,

"limit": 20,

"receiver": {

"id": "522a6cd6e4b0a7a09b1ff098",

"name": "mimi"

}

}

The searchmessage.py command script was used above to retrieve unread messages for mimi. The search result returns a page of messages using offset and limit to walk through the available messages. (As noted above the search and paging should be based off of a baseline timestamp in order to be stable with concurrent message activity.)

Having read the message, mimi now sets the status to “read”.

Gregorys-MacBook-Pro:python greg$ **python updatemessagestatus.py -l mimi -p secret -i 522a6d0ae4b0a7a09b1ff09a -s read -v**

########## request: http://ec2-54-200-9-5.us-west-2.compute.amazonaws.com:8080/message/rest/user/name/mimi

########## request: http://ec2-54-200-9-5.us-west-2.compute.amazonaws.com:8080/message/rest/message/522a6d0ae4b0a7a09b1ff09a

{u'status': u'read', u'sender': {u'id': u'522a6cc4e4b0a7a09b1ff097', u'name': u'greg'}, u'content': u'hello world', u'receiver': {u'id': u'522a6cd6e4b0a7a09b1ff098', u'name': u'mimi'}, u'id': u'522a6d0ae4b0a7a09b1ff09a', u'createdAt': 1378512138372}

Searching for “unread” messages now shows none. It’s also easy to see that search sends the original search criteria back with the search results.

Gregorys-MacBook-Pro:python greg$ **python searchmessages.py -l mimi -p secret -v -s unread**

########## request: http://ec2-54-200-9-5.us-west-2.compute.amazonaws.com:8080/message/rest/user/name/mimi

########## request: http://ec2-54-200-9-5.us-west-2.compute.amazonaws.com:8080/message/rest/message/search?status=unread&receiverId=522a6cd6e4b0a7a09b1ff098&limit=20&offset=0

>>>>>>>>>> search results

{

"status": "unread",

"messages": [],

"offset": 0,

"limit": 20,

"receiver": {

"id": "522a6cd6e4b0a7a09b1ff098",

"name": "mimi"

}

}

Searching for “read” messages shows one:

Gregorys-MacBook-Pro:python greg$ **python searchmessages.py -l mimi -p secret -v -s read**

########## request: http://ec2-54-200-9-5.us-west-2.compute.amazonaws.com:8080/message/rest/user/name/mimi

########## request: http://ec2-54-200-9-5.us-west-2.compute.amazonaws.com:8080/message/rest/message/search?status=read&receiverId=522a6cd6e4b0a7a09b1ff098&limit=20&offset=0

>>>>>>>>>> search results

{

"status": "read",

"messages": [

{

"status": "read",

"sender": {

"id": "522a6cc4e4b0a7a09b1ff097",

"name": "greg"

},

"content": "hello world",

"receiver": {

"id": "522a6cd6e4b0a7a09b1ff098",

"name": "mimi"

},

"id": "522a6d0ae4b0a7a09b1ff09a",

"createdAt": 1378512138372

}

],

"offset": 0,

"limit": 20,

"receiver": {

"id": "522a6cd6e4b0a7a09b1ff098",

"name": "mimi"

}

}

## Creating Groups and Group Messaging

Create a group for “students”.

Gregorys-MacBook-Pro:python greg$ **python creategroup.py -l greg -p secret -g students**

{u'id': u'522a6dffe4b0a7a09b1ff09b', u'createdAt': 1378512383265, u'name': u'students'}

Create some users and add them to the group.

Gregorys-MacBook-Pro:python greg$ **python creategroup.py -l greg -p secret -g students**

{u'id': u'522a6dffe4b0a7a09b1ff09b', u'createdAt': 1378512383265, u'name': u'students'}

Gregorys-MacBook-Pro:python greg$ python createuser.py -l bob1 -p secret

{u'loginName': u'bob1', u'createdAt': 1378512451759, u'id': u'522a6e43e4b0a7a09b1ff09c', u'groupRefs': []}

Gregorys-MacBook-Pro:python greg$ **python joingroup.py -l bob1 -p secret -g students**

{u'loginName': u'bob1', u'createdAt': 1378512451759, u'id': u'522a6e43e4b0a7a09b1ff09c', u'groupRefs': [{u'id': u'522a6dffe4b0a7a09b1ff09b', u'name': u'students'}]}

Gregorys-MacBook-Pro:python greg$ **python createuser.py -l bob2 -p secret; python joingroup.py -l bob2 -p secret -g students**

{u'loginName': u'bob2', u'createdAt': 1378512581627, u'id': u'522a6ec5e4b0a7a09b1ff09d', u'groupRefs': []}

{u'loginName': u'bob2', u'createdAt': 1378512581627, u'id': u'522a6ec5e4b0a7a09b1ff09d', u'groupRefs': [{u'id': u'522a6dffe4b0a7a09b1ff09b', u'name': u'students'}]}

Gregorys-MacBook-Pro:python greg$ **python createuser.py -l mark -p secret; python joingroup.py -l mark -p secret -g students**

{u'loginName': u'mark', u'createdAt': 1378512636537, u'id': u'522a6efce4b0a7a09b1ff09e', u'groupRefs': []}

{u'loginName': u'mark', u'createdAt': 1378512636537, u'id': u'522a6efce4b0a7a09b1ff09e', u'groupRefs': [{u'id': u'522a6dffe4b0a7a09b1ff09b', u'name': u'students'}]}

Gregorys-MacBook-Pro:python greg$ **python createuser.py -l gerald -p secret; python joingroup.py -l gerald -p secret -g students**

{u'loginName': u'gerald', u'createdAt': 1378512680913, u'id': u'522a6f28e4b0a7a09b1ff09f', u'groupRefs': []}

{u'loginName': u'gerald', u'createdAt': 1378512680913, u'id': u'522a6f28e4b0a7a09b1ff09f', u'groupRefs': [{u'id': u'522a6dffe4b0a7a09b1ff09b', u'name': u'students'}]}

Gregorys-MacBook-Pro:**python greg$ python createuser.py -l alan -p secret; python joingroup.py -l alan -p secret -g students**

{u'loginName': u'alan', u'createdAt': 1378512741514, u'id': u'522a6f65e4b0a7a09b1ff0a0', u'groupRefs': []}

{u'loginName': u'alan', u'createdAt': 1378512741514, u'id': u'522a6f65e4b0a7a09b1ff0a0', u'groupRefs': [{u'id': u'522a6dffe4b0a7a09b1ff09b', u'name': u'students'}]}

Send a message to the students group:

Gregorys-MacBook-Pro:python greg$ **python createmessage.py -g students -p secret -l greg -c "are we not men?" -s send**

{u'status': u'sent', u'group': {u'id': u'522a6dffe4b0a7a09b1ff09b', u'name': u'students'}, u'sender': {u'id': u'522a6cc4e4b0a7a09b1ff097', u'name': u'greg'}, u'content': u'are we not men?', u'id': u'522a6fcfe4b0a7a09b1ff0a1', u'createdAt': 1378512847345}

Have mark read his messages:

Gregorys-MacBook-Pro:python greg$ **python searchmessages.py -l mark -p secret -v -s unread**

########## request: http://ec2-54-200-9-5.us-west-2.compute.amazonaws.com:8080/message/rest/user/name/mark

########## request: http://ec2-54-200-9-5.us-west-2.compute.amazonaws.com:8080/message/rest/message/search?status=unread&receiverId=522a6efce4b0a7a09b1ff09e&limit=20&offset=0

>>>>>>>>>> search results

{

"status": "unread",

"messages": [

{

"status": "unread",

"group": {

"id": "522a6dffe4b0a7a09b1ff09b",

"name": "students"

},

"sender": {

"id": "522a6cc4e4b0a7a09b1ff097",

"name": "greg"

},

"content": "are we not men?",

"receiver": {

"id": "522a6efce4b0a7a09b1ff09e",

"name": "mark"

},

"id": "522a6fcfe4b0a7a09b1ff0a4",

"createdAt": 1378512847345

}

],

"offset": 0,

"limit": 20,

"receiver": {

"id": "522a6efce4b0a7a09b1ff09e",

"name": "mark"

}

}

Mark marks the message as “read”.

Gregorys-MacBook-Pro:python greg$ **python updatemessagestatus.py -l mark -p secret -i 522a6fcfe4b0a7a09b1ff0a4 -s read**

{u'status': u'read', u'group': {u'id': u'522a6dffe4b0a7a09b1ff09b', u'name': u'students'}, u'sender': {u'id': u'522a6cc4e4b0a7a09b1ff097', u'name': u'greg'}, u'content': u'are we not men?', u'receiver': {u'id': u'522a6efce4b0a7a09b1ff09e', u'name': u'mark'}, u'id': u'522a6fcfe4b0a7a09b1ff0a4', u'createdAt': 1378512847345}

Bob1 checks his inbox:

Gregorys-MacBook-Pro:python greg$ **python searchmessages.py -l bob1 -p secret -v -s unread**

########## request: http://ec2-54-200-9-5.us-west-2.compute.amazonaws.com:8080/message/rest/user/name/bob1

########## request: http://ec2-54-200-9-5.us-west-2.compute.amazonaws.com:8080/message/rest/message/search?status=unread&receiverId=522a6e43e4b0a7a09b1ff09c&limit=20&offset=0

>>>>>>>>>> search results

{

"status": "unread",

"messages": [

{

"status": "unread",

"group": {

"id": "522a6dffe4b0a7a09b1ff09b",

"name": "students"

},

"sender": {

"id": "522a6cc4e4b0a7a09b1ff097",

"name": "greg"

},

"content": "are we not men?",

"receiver": {

"id": "522a6e43e4b0a7a09b1ff09c",

"name": "bob1"

},

"id": "522a6fcfe4b0a7a09b1ff0a2",

"createdAt": 1378512847345

}

],

"offset": 0,

"limit": 20,

"receiver": {

"id": "522a6e43e4b0a7a09b1ff09c",

"name": "bob1"

}

}

Here we can see that there are separate instances of the message for each recpient. Mark’s instance of the message has a different unique id than Bob’s instance of the message. In addition to control the lifecycle of their own messages independently, this design lends itself to database sharding of the message by user and this is what makes it possible to scale to large numbers of users.

Exercise left to the reader: You check the inbox for mimi and you will see that she did not receive the email that went to the students.

## Sending a message to all users:

To send a message to all users, use the “all” group. Note however that users do not need, nor are they, explicitly members of the all group.

**python createmessage.py -g all -p secret -l greg -c "hello everyone" -s send**