#contentWrapper #fs, #sidebarContent #fs, #contentWrapper div [id * = 'myExtraContent'], #sidebarContent div [id * = 'myExtraContent'] {display: block;}

Kermith's workshop (https://translate.googleusercontent.cc depth=1&hl=en&prev=search&pto=aue&rurl=translate.goog

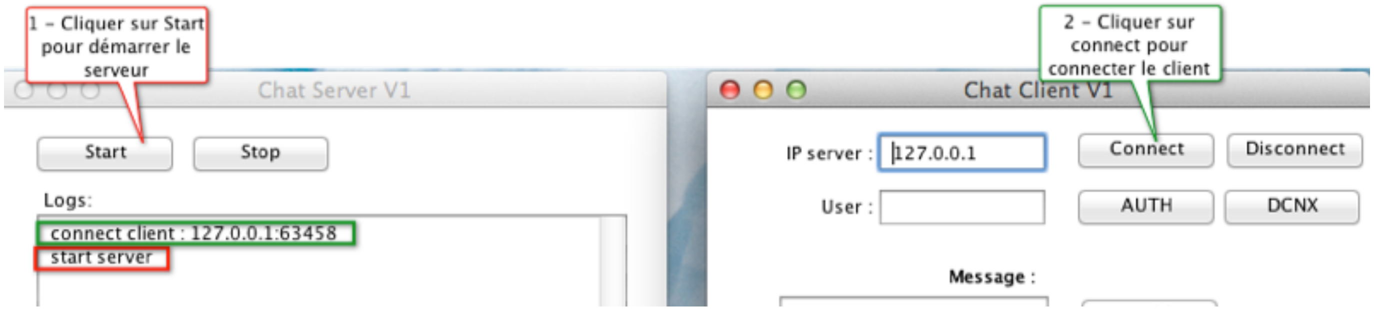
The other way to see supervision ...

Create a chat server

Today we will find out how to make a Chat server. This article is taken from LiveCode tutorials (https://translate.googleusercontent.com/translate_c?depth=1&hl=en&prev=search&pto=aue&rurl=translate.google.com&sl=fr&sp=nmt4&u=http://lesso how-to-communicate-with-other-applications-using-sockets&usg=ALkJrhhzbympnI71eKkDC9k72B6Sk_ncFA) . First, as the article indicates, we will start to develop a communication protocol:
- we must check the connections of the computers,
- then, we must accept the identifier (user) and check that it is unique ,
- the messages will be processed and automatically returned to all connected users,
- we will deal with the case of the disconnection of a user,
- the complete disconnection of the client,
- and of course the processing of errors.

Sequence analysis

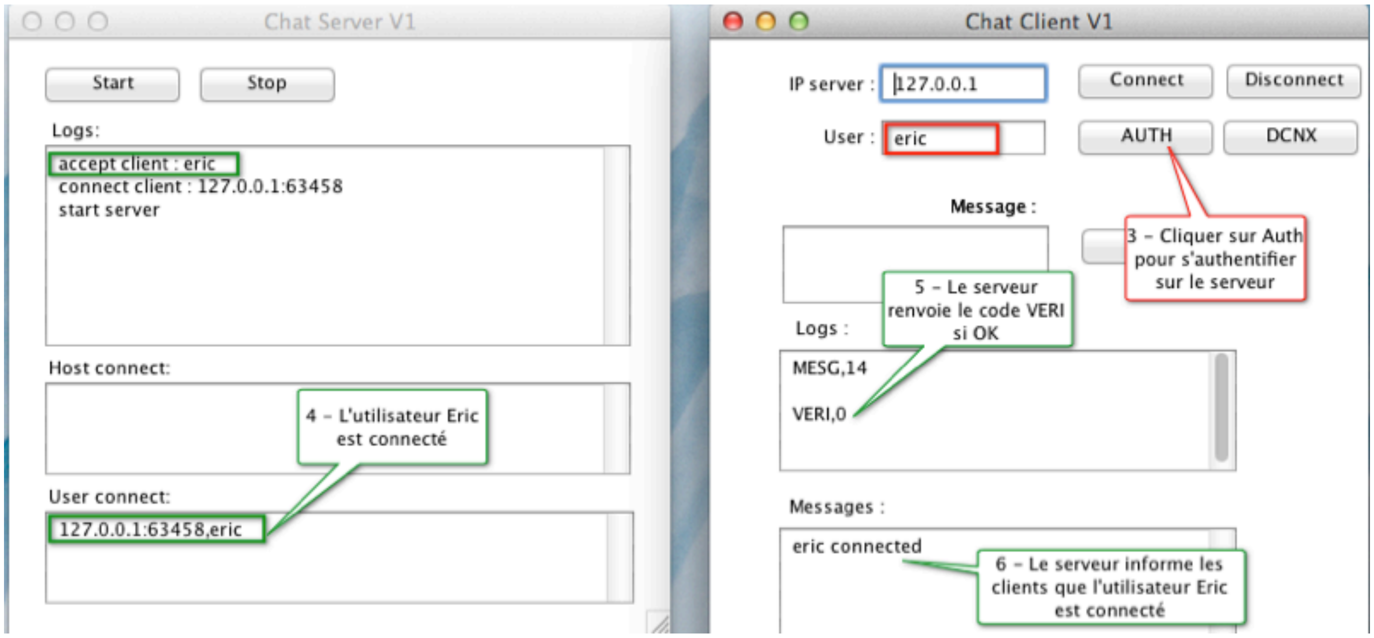
Here is a series of screenshots explaining a sequence of connection of a client on our chat server.


1 - Cliquer sur Start pour démarrer le serveur
2 - Cliquer sur connect pour connecter le client
Chat Server V1
Start
Stop
Logs:
Chat Client V1
IP server :
User :
Connect
Disconnect
AUTH
DCNX
Message :

```
connect client : 127.0.0.1:63458
start server
```

IP server : 127.0.0.1

1 - Customer connection


Chat Server V1
Start
Stop
Logs:
Host connect:
User connect:
Chat Client V1
IP server :
Connect
Disconnect
User :
AUTH
DCNX
Message :
Logs :
Messages :
3 – Cliquer sur Auth pour s'authentifier sur le serveur
4 – L'utilisateur Eric est connecté
5 – Le serveur renvoie le code VERI si OK
6 – Le serveur informe les clients que l'utilisateur Eric est connecté

```
accept client : eric
connect client : 127.0.0.1:63458
start server
```

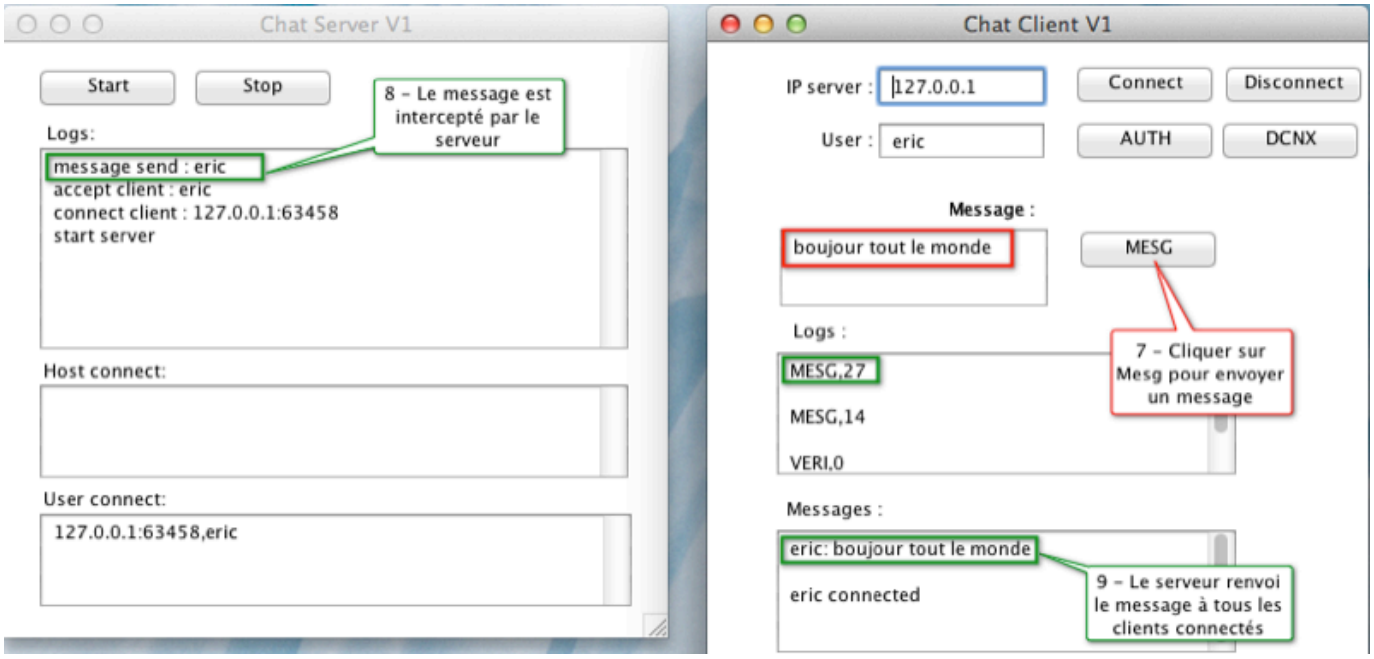
```
127.0.0.1:63458,eric
```

IP server : 127.0.0.1
User : eric

```
MESG,14

VERI,0
```

```
eric connected
```

2 - User authentication


Chat Server V1
Start
Stop
Logs:
Host connect:
User connect:
Chat Client V1
IP server :
Connect
Disconnect
User :
AUTH
DCNX
Message :
MESG
Logs :
Messages :
7 – Cliquer sur Mesg pour envoyer un message
8 – Le message est intercepté par le serveur
9 – Le serveur renvoi le message à tous les clients connectés

```
message send : eric
accept client : eric
connect client : 127.0.0.1:63458
start server
```

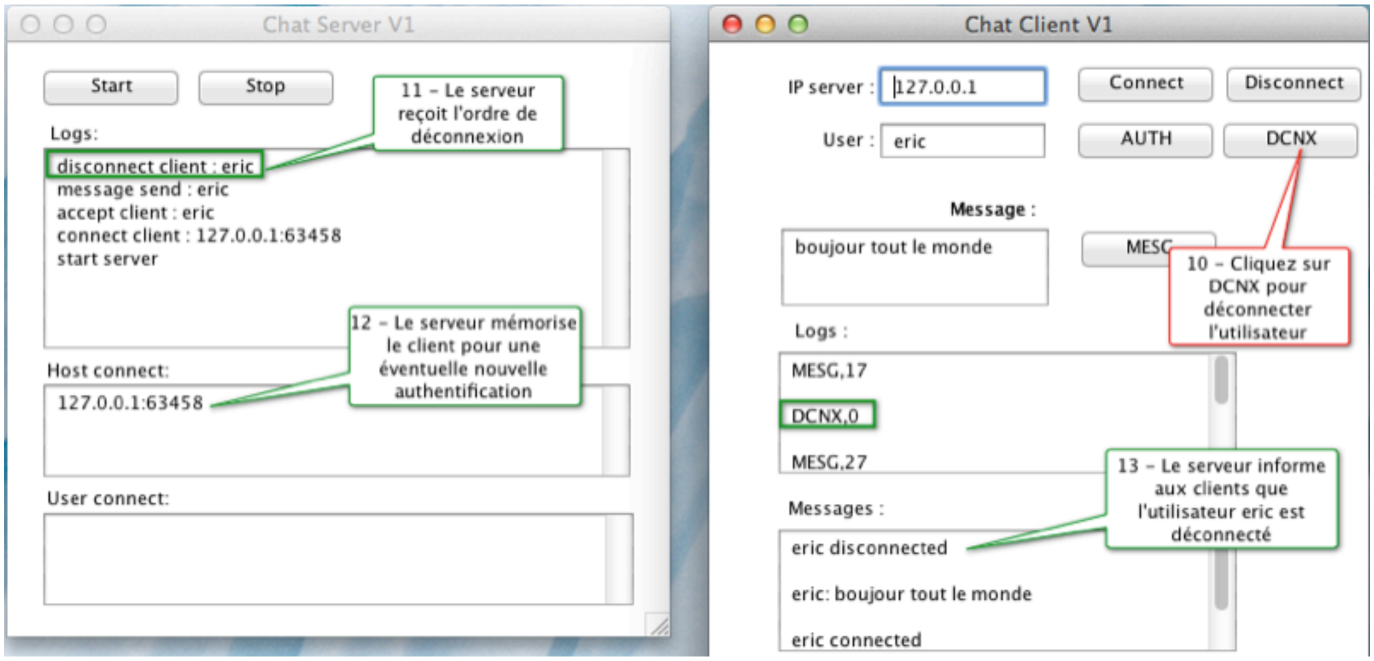
```
127.0.0.1:63458,eric
```

IP server : 127.0.0.1
User : eric
Message : boujour tout le monde

```
MESG,27

MESG,14

VERI,0
```

```
eric: boujour tout le monde

eric connected
```

3 - Sending the message



```
disconnect client : eric
message send : eric
accept client : eric
connect client : 127.0.0.1:63458
start server
```

Host connect: 127.0.0.1:63458

IP server : 127.0.0.1
User : eric
Message : boujour tout le monde

```
MESG,17
DCNX,0
MESG,27
```

```
eric disconnected
eric: boujour tout le monde
eric connected
```

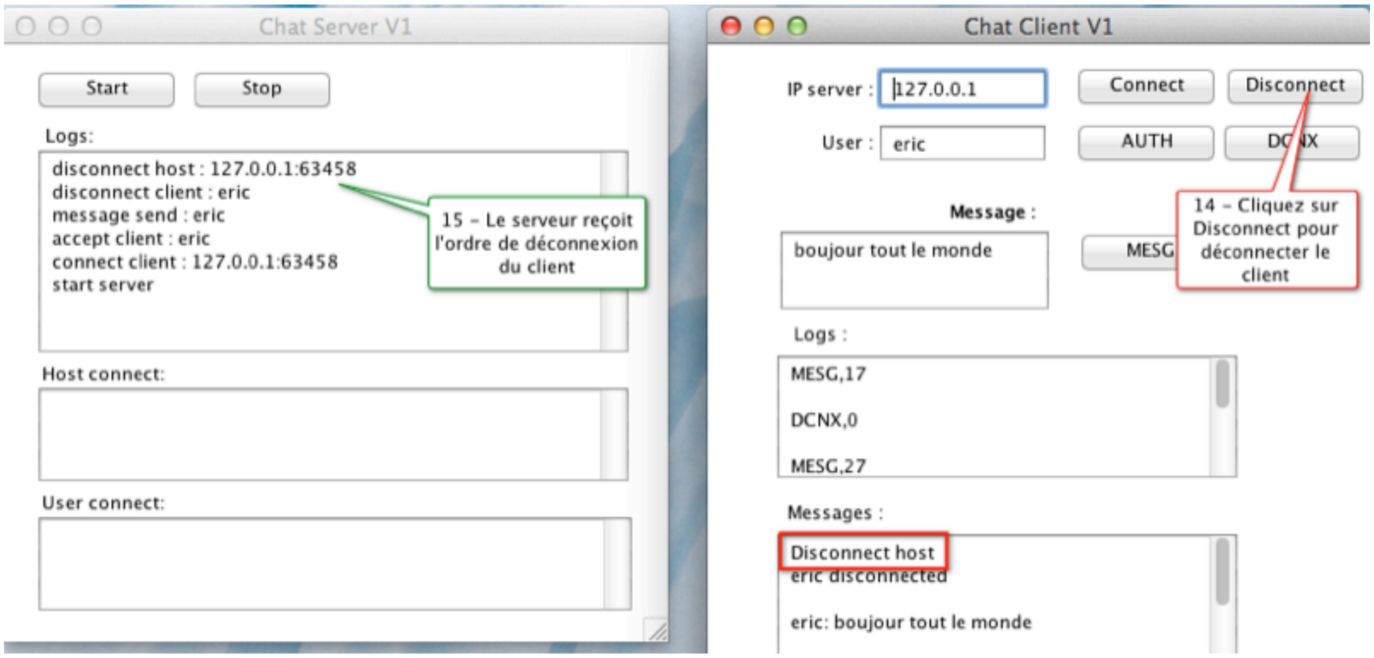
11 – Le serveur reçoit l'ordre de déconnexion

12 – Le serveur mémorise le client pour une éventuelle nouvelle authentification

10 – Cliquez sur DCNX pour déconnecter l'utilisateur

13 – Le serveur informe aux clients que l'utilisateur eric est déconnecté

4 - User logout



```
disconnect host : 127.0.0.1:63458
disconnect client : eric
message send : eric
accept client : eric
connect client : 127.0.0.1:63458
start server
```

IP server : 127.0.0.1
User : eric
Message : boujour tout le monde

```
MESG,17
DCNX,0
MESG,27
```

```
Disconnect host
eric disconnected
eric: boujour tout le monde
```

15 – Le serveur reçoit l'ordre de déconnexion du client

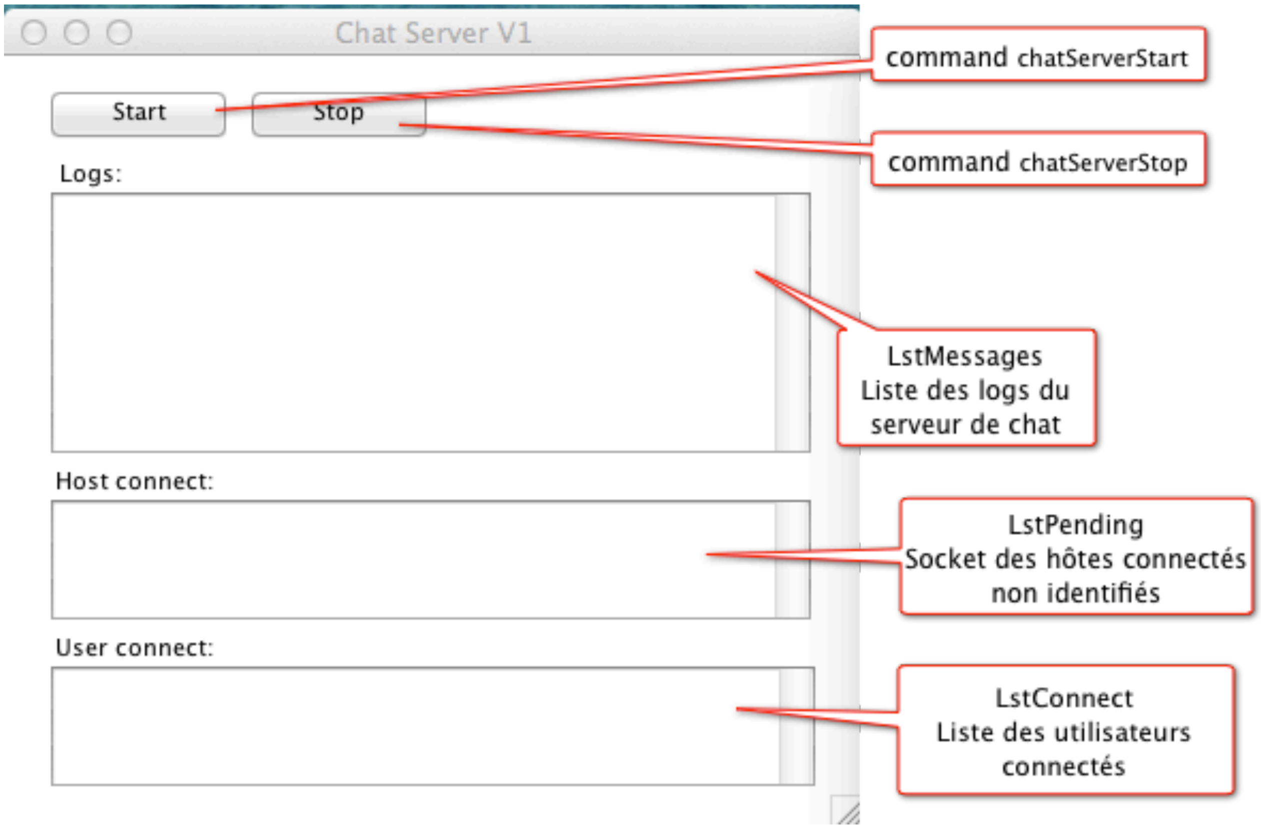
14 – Cliquez sur Disconnect pour déconnecter le client

5 - Customer disconnection

The program

The programs were carried out for educational purposes. I deliberately omitted the establishment of a mechanism to control the sequence of sequences. You can perform illegal operations like sending messages without user authentication to check for error handling.

Server program



I indicate the procedures located in the handler of the card for the buttons

Le programme est relativement simple, tout le code se situe principalement dans le handler de la carte. Les boutons font appel aux procédures chatServerStart et chatServerStop.

```
local sConnectedClients    -- Liste des utilisateurs authorisés [utilisateur] => [nom]
local sPendingClients      -- liste des hôtes en attente
local sClientNames         -- Nom de l'utilisateur courant
local sRunning             -- serveur en cours
constant kPort = 8020

-- Démarre le serveur
command chatServerStart
  if not sRunning then
    put true into sRunning
    put empty into field "LstMessages"
    put empty into field "LstPending"
    put empty into field "LstConnect"
    put "start server" & return before field "LstMessages"
    accept connections on port kPort with message "chatServerClientConnected"
  end if
end chatServerStart
```

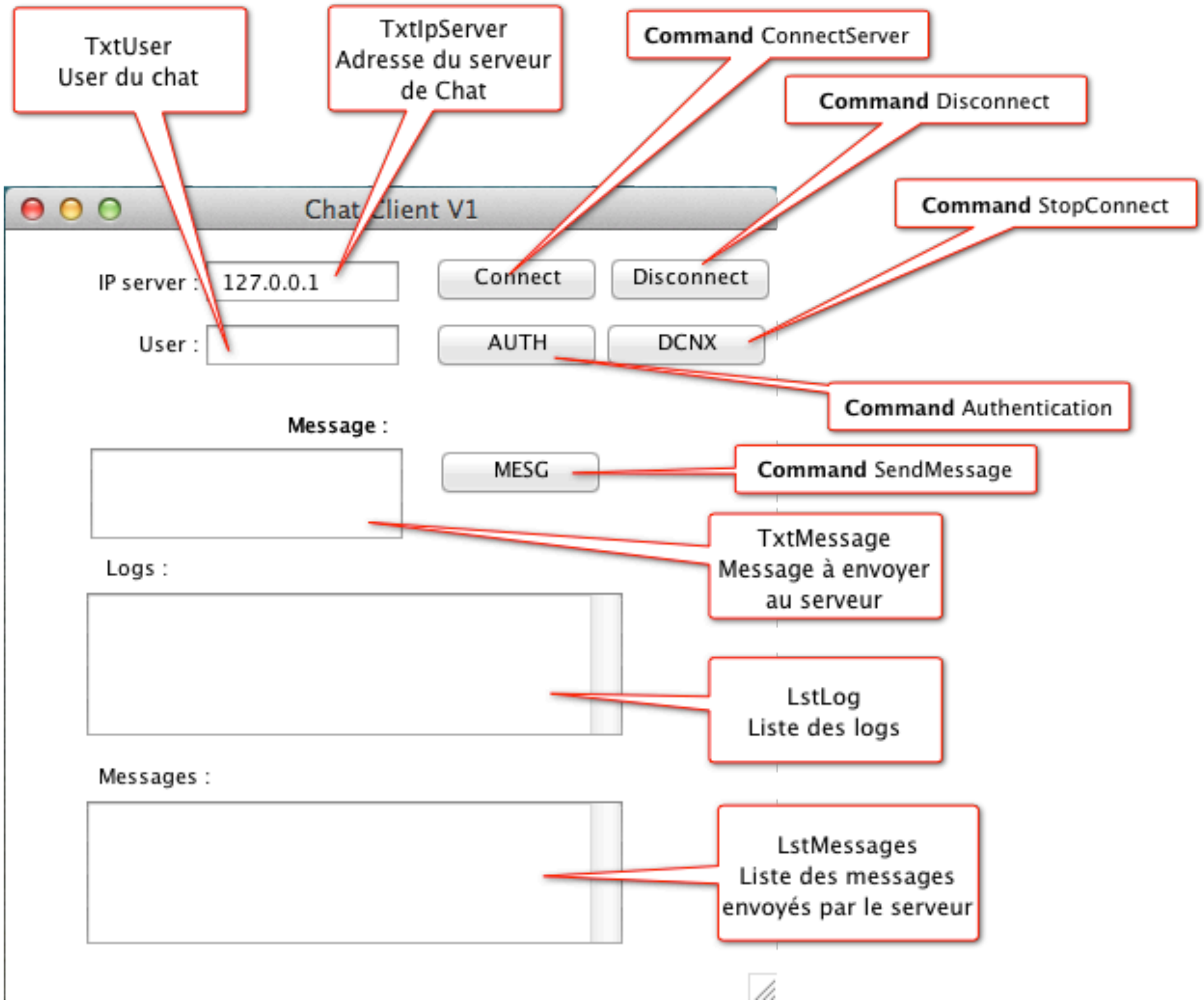
```
-- Stoppe le serveur
command chatServerStop
  if sRunning then
    put false into sRunning
    put empty into sConnectedClients
    put empty into sPendingClients
    put empty into sClientNames
    put empty into field "LstPending"
    put empty into field "LstConnect"
    repeat for each line tSocket in the opensockets
      close socket tSocket
      put "deconnect socket : " & tSocket & return before field "LstMessages"
    end repeat
    put "stop server" & return before field "LstMessages"
  end if
end chatServerStop

on chatServerClientConnected pSocket
  put pSocket & return after sPendingClients
  put sPendingClients into field "LstPending"
  put "connect client : " & pSocket & return before field "LstMessages"
  read from socket pSocket until return with message "chatServerMessageReceived"
end chatServerClientConnected

on chatServerMessageReceived pSocket, pMsg
  if length(pMsg) > 1 then
    put char 1 to -2 of pMsg into pMsg
    local tAuth, tCommand, tLength, tMsg
    put pSocket is among the keys of sConnectedClients into tAuth
    put item 1 of pMsg into tCommand
    put item 2 of pMsg into tLength
    if tLength is not an integer then
      put "Invalid message length" & return into tMsg
      write "WARN," & the number of chars in tMsg & return & tMsg & return to socket
pSocket
    else
      switch tCommand
    case "DCNX"
    -- Déconnexion de l'utilisateur
    if tAuth then
      read from socket pSocket for tLength chars
      if it is among the lines of sClientNames then
        put "disconnect client : " & it & return before field "LstMessages"
        write "DCNX,0" & return to socket pSocket
        chatServerBroadcast it && "disconnected"

        delete line lineoffset(it, sClientNames) of sClientNames
        put pSocket & return after sPendingClients
        put sPendingClients into field "LstPending"
        put empty into sConnectedClients[pSocket]
        delete line lineoffset(pSocket, sConnectedClients) of sConnectedClients
        put empty into field "LstConnect"
        repeat for each line tSocket in the keys of sConnectedClients
          put tSocket & "," & sConnectedClients[tSocket] into field "LstConnect"
        end repeat
      end if
    else
      put "Client not verified" & return into tMsg
      write "ERRO," & the number of chars in tMsg & return & tMsg to socket pSocket
    end if
    break
  case "STOP"
    -- Déconnexion de l'hôte
    if tAuth then
      -- l'utilisateur est connecté, deconnexion automatique
      read from socket pSocket for tLength chars
      if it is among the lines of sClientNames then
        put "disconnect client : " & it & return before field "LstMessages"
        delete line lineoffset(pSocket, sConnectedClients) of sConnectedClients
        write "DCNX,0" & return to socket pSocket
        chatServerBroadcast it && "disconnected"
        delete line lineoffset(it, sClientNames) of sClientNames
      end if
      put empty into field "LstConnect"
      repeat for each line tSocket in the keys of sConnectedClients
```

```
          put tSocket & "," & sConnectedClients[tSocket] into field "LstConnect"
        end repeat

      end if
      if pSocket is among the lines of sPendingClients then
        delete line lineoffset(pSocket, sPendingClients) of sPendingClients
      end if
      put "disconnect host : " & pSocket & return before field "LstMessages"
      put sPendingClients into field "LstPending"

      break
    case "MESG"
      -- gestion des messages
      if tAuth then
        read from socket pSocket for tLength chars
        put "message send : " & sConnectedClients[pSocket] & return before field
"LstMessages"
        chatServerBroadcast sConnectedClients[pSocket] & ":" && it
      else
        put "Client not verified" & return into tMsg
        write "ERRO," & the number of chars in tMsg & return & tMsg to socket pSocket
       end if
      break
    case "AUTH"
      -- authentification de l'utilisateur
      if tAuth then
        put "Client already verified" & return into tMsg
        write "WARN," & the number of chars in tMsg & return & tMsg to socket
pSocket
      else
        read from socket pSocket for tLength chars
        if it is not among the lines of sClientNames then
          put it into sConnectedClients[pSocket]
          put "accept client : " & it & return before field "LstMessages"
          put it & return after sClientNames
          write "VERI,0" & return to socket pSocket
          delete line lineoffset(pSocket, sPendingClients) of sPendingClients
          put sPendingClients into field "LstPending"
```

```
            repeat for each line tSocket in the keys of sConnectedClients
              put tSocket & "," & sConnectedClients[tSocket] into field "LstConnect"
            end repeat
            chatServerBroadcast it && "connected"
          else
            put "Username already taken" & return into tMsg
            write "ERRO," & the number of chars in tMsg & return & tMsg to socket pSocket
          end if
        end if
        break
      default
        put "Unknown command" & return into tMsg
        write "ERRO," & the number of chars in tMsg & return & tMsg to socket pSocket
        break
      end switch
    end if
  end if
  read from socket pSocket until return with message "chatServerMessageReceived"
end chatServerMessageReceived

command chatServerBroadcast pMsg
  local tMsg
  put "MESG," & the number of chars in pMsg & return & pMsg & return into tMsg
  repeat for each line tSocket in the keys of sConnectedClients
    write tMsg to socket tSocket
  end repeat
end chatServerBroadcast

on socketClosed pSocket
  if pSocket is among the lines of sPendingClients then
    delete line lineoffset(pSocket, sPendingClients) of sPendingClients
  else if sConnectedClients[pSocket] is not empty then
    local tName
    put sConnectedClients[pSocket] into tName
    delete variable sConnectedClients[pSocket]
    delete line lineoffset(tName, sClientNames) of sClientNames
    chatServerBroadcast tName && "disconnected"
  end if
end socketClosed
```

Le code principal se trouve dans la procédure *chatServerMessageReceived* initialisé par la commande *read from socket.*

Programme client



J'indique les procédures situées dans l'handler de la carte pour les boutons

Le programme du client est encore plus simple, tout le code se situe principalement dans le handler de la carte.

```
local sIpServer,sSocket
constant kPort = 8020

Command ConnectServer
```

```
  put field "TxtIpServer" into sIpServer
  put empty into field "LstMessages"
  put empty into field "LstLog"
  open socket to sIpServer & ":" & kPort with message "ClientConnect"
end ConnectServer

Command Disconnect
  local tUser,tlengthUser
  put field "TxtUser" into tUser
  put the length of tUser into tlengthUser
  write "STOP," & tlengthUser & return & tuser & return to socket sIpServer & ":" & kPort
  put "Disconnect host" & return before field "LstMessages"
  close socket sSocket
end Disconnect

Command Authentication
  local tUser,tlengthUser
  put field "TxtUser" into tUser
  put the length of tUser into tlengthUser
  if tlengthUser > 0 then
    write "AUTH," & tlengthUser & return & tuser & return to socket sIpServer & ":" & kPort
  else
    put "Error authentication" & return before field "LstMessages"
  end if
end Authentication

Command StopConnect
  local tUser,tlengthUser
  put field "TxtUser" into tUser
  put the length of tUser into tlengthUser
  write "DCNX," & tlengthUser & return & tuser & return to socket sIpServer & ":" & kPort
end StopConnect

Command SendMessage
  local tMessage, tlenghtMessage
  put field "TxtMessage" into tMessage
  put the length of tMessage into tlengthMessage
  write "MESG," & tlengthMessage & return & tMessage & return to socket sIpServer & ":" &
```

```
kPort
end SendMessage

Command ClientConnect pSocket
  put pSocket into sSocket
  read from socket sSocket until return with message "ClientConnectReceveid"
end ClientConnect

Command ClientConnectReceveid pSocket, pMesg
  local tCommand, tLength
  put item 1 of pMesg into tCommand
  put item 2 of pMesg into tLength
  if tLength is an integer then
    put pMesg & return before field "LstLog"
  else
    put pMesg & return before field "LstMessages"
  end if
  read from socket sSocket until return with message "ClientConnectReceveid"
end ClientConnectReceveid
```

Les sources

- Le programme client (https://translate.googleusercontent.com/translate_c?depth=1&hl=en&prev=search&pto=aue&rurl=translate.google.com&sl=fr&sp=nmt4&u=https://w
Client-V1.livecode&usg=ALkJrhiuBYYtlpCUXvNIjs3Ryy-G-7a89w),
- le programme serveur (https://translate.googleusercontent.com/translate_c?depth=1&hl=en&prev=search&pto=aue&rurl=translate.google.com&sl=fr&sp=nmt4&u=https://w
Server-V1.livecode&usg=ALkJrhhQptyVN2JVPa0ypdiOLX2CpHqjXw).

0 Commentaires Sugarbug Règles de confidentialité de Disqus 1 S'identifier

Recommander Tweet Partager Les meilleurs

Commencer la discussion…

S'IDENTIFIER AVEC

OU INSCRIVEZ-VOUS SUR DISQUS ?

Nom

Soyez le premier à commenter.

S'abonner Ajoutez Disqus à votre site web !Ajouter DisqusAjouter