The Kermith workshop (https://translate.googleusercontent.com/translate_c?depth=1&hl=en&prev=search&rurl=translate.google.com&sl=fr&sp=nmt4&u=http://www.sugarbu

The other way to see supervision ...

Make a Chat Server

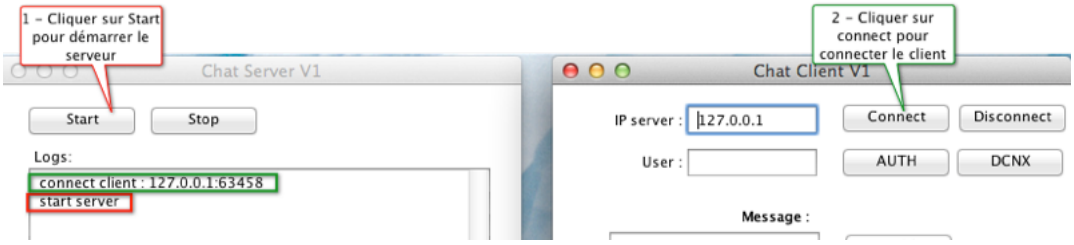
Today we will discover how to build a Chat Server. This article is from the LiveCode tutorials (https://translate.googleusercontent.com/translate_c?depth=1&hl=en&prev=search&rurl=translate.google.com&sl=fr&sp=nmt4&u=http://lessons.runrev.com/s/lessons/m/4071/l/12924-how-to-communicate-with-other-applications-using-sockets&xid=25657,15700022,15700124,15700149,15700186,15700191,15700201&usg=ALkJrhiSRVHDe7bBCud4_mmmDDXPGJdmeA) .

First, as the article states, we will begin to develop a communication protocol:

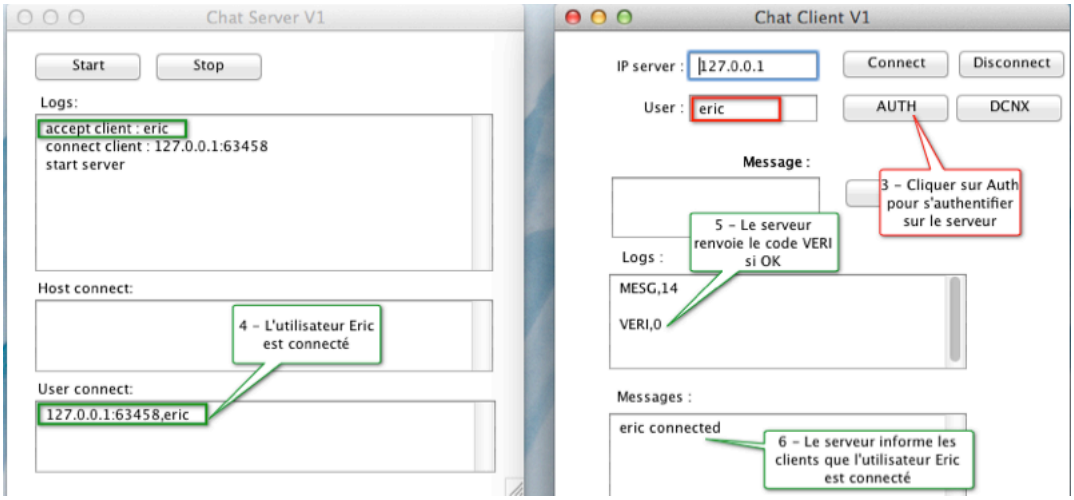
- we have to check the connections of the computers,
- then, you have to accept the identifier (user) and check that it is unique,
- messages will be processed and automatically sent back to all connected users,
- we will deal with the case of the disconnection of a user,
- the complete disconnection of the customer,
- and of course the treatment of errors.

Analysis of a sequence

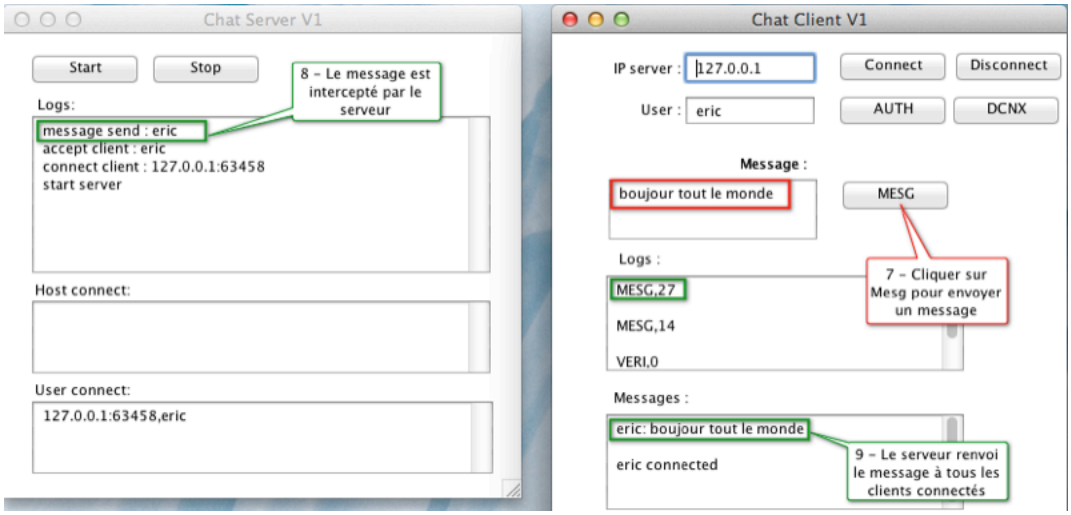
Here is a series of screenshots explaining a connection sequence of a client on our chat server.



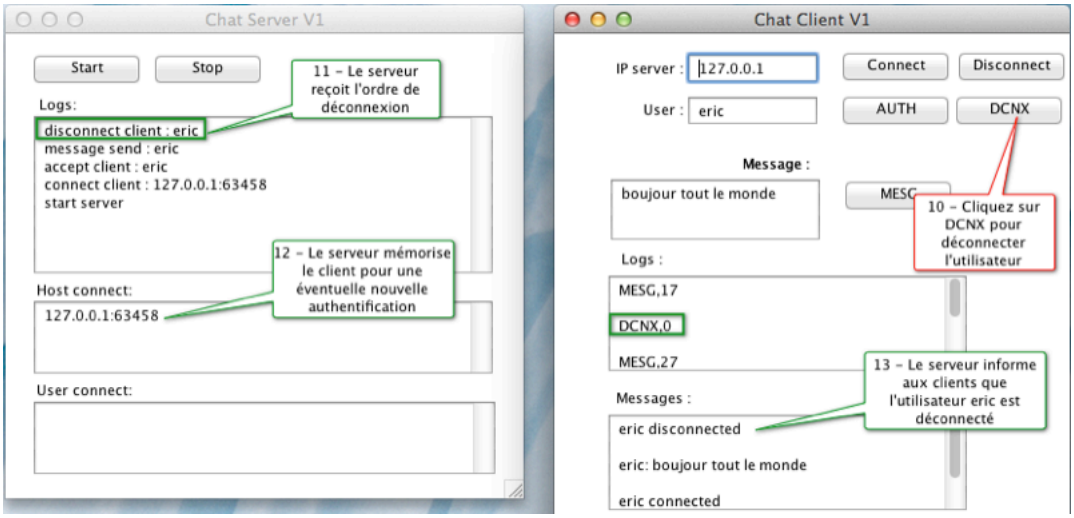
1 - Customer login



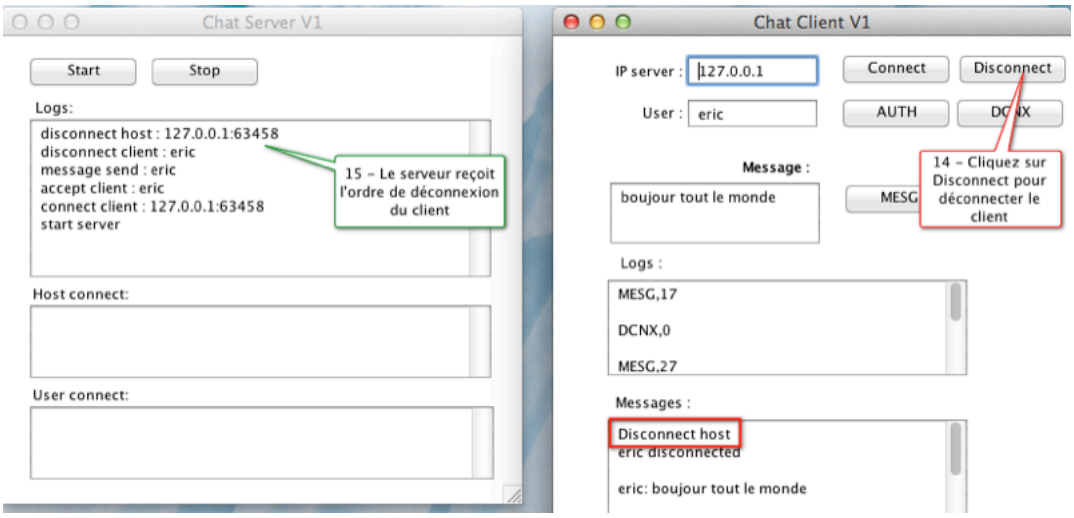
2 - User Authentication



3 - Sending the message



4 - User disconnection

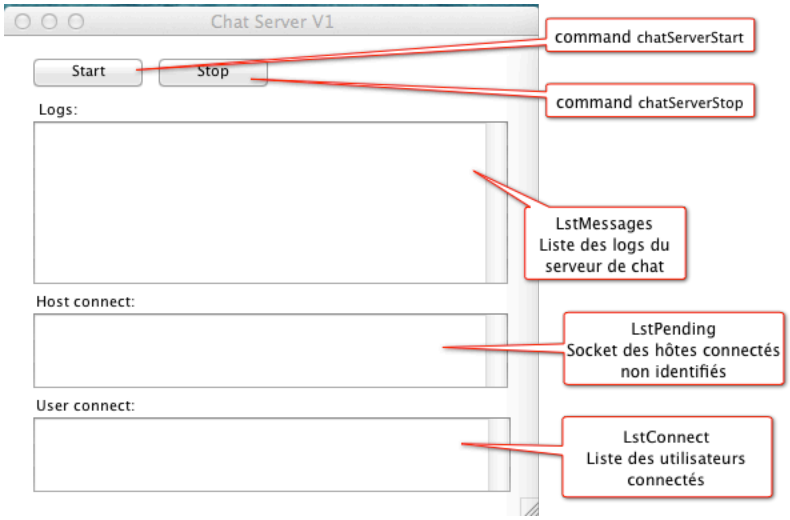


5 - Customer disconnection

The program

The programs were made for didactic purposes. I have deliberately omitted the establishment of a mechanism for controlling the sequence of sequences. You can perform unauthorized operations such as sending messages without user authentication to verify error handling.

Server program



I indicate the procedures located in the map handler for buttons

The program is relatively simple, all the code is mainly in the handler of the card. The buttons use the chatServerStart and chatServerStop procedures.

```
local sConnectedClients - List of Authorized Users [user] => [name]
local sPendingClients - list of pending hosts
local sClientNames - Name of the current user
local sRunning - server in progress
constant kPort = 8020

- Start the server
command chatServerStart
   yew not sRunning then
      could true into sRunning
      could empty into field "LstMessages"
      could empty into field "LstPending"
      could empty into field "LstConnect"
      could "start server" & return before field "LstMessages"
      accept connections we port kPort with message "chatServerClientConnected"
   end yew
end chatServerStart

- Stop the server
command chatServerStop
   if sRunning then
      could false into sRunning
      could empty into sConnectedClients
      could empty into sPendingClients
      could empty into sClientNames
      could empty into field "LstPending"
      could empty into field "LstConnect"
      repeat for Each line tSocket in tea opensockets
         close tSocket socket
         could "deconnect socket:" & tSocket & return before field "LstMessages"
      end repeat
      could "stop server" & return before field "LstMessages"
   end yew
end chatServerStop

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

on chatServerMessageReceived pSocket, pMsg
   yew length (pMsg)> 1 Then
      could tank 1 to - 2 of pMsg into pMsg
      local tAuth, tCommand, tLength, tMsg
```

```
put pSocket is Among tea keys of sConnectedClients into tAuth
could item 1 of pMsg into tCommand
could item 2 of pMsg into tLength
if tLength is not year integer Then
  could "Invalid message length" & return into tMsg
  write "WARN," & the number of tanks in tMsg & return & tMsg & return to pSocket socket
else
  switch tCommand
    hut "DCNX"
    - User disconnection
    if tAuth then
      read from socket pSocket for tLength tanks
      if it is Among tea lines of sClientNames then
        could "client disconnect:" & it & return before field "LstMessages"
        write "DCNX, 0" & return to pSocket socket
        chatServerBroadcast it && "disconnected"

        delete line lineoffset (it, sClientNames) of sClientNames
        put pSocket & return after sPendingClients
        put sPendingClients into field "LstPending"
        could empty into sConnectedClients [pSocket]
        delete line lineoffset (pSocket, sConnectedClients) of sConnectedClients
        could empty into field "LstConnect"
        repeat for Each line tSocket in tea keys of sConnectedClients
          put tSocket & "," & sConnectedClients [tSocket] into field "LstConnect"
        end repeat
      end yew
    else
      could "Client not verified" & return into tMsg
      write "ERRO," & the number of tanks in tMsg & return & tMsg to pSocket socket
    end yew
    break
  hut "STOP"
    - Disconnecting the host
    if tAuth then
      - the user is logged in, automatic logout
      read from socket pSocket for tLength tanks
      if it is Among tea lines of sClientNames then
        could "client disconnect:" & it & return before field "LstMessages"
        delete line lineoffset (pSocket, sConnectedClients) of sConnectedClients
        write "DCNX, 0" & return to pSocket socket
        chatServerBroadcast it && "disconnected"
        delete line lineoffset (it, sClientNames) of sClientNames
      end yew
      could empty into field "LstConnect"
      repeat for Each line tSocket in tea keys of sConnectedClients
        put tSocket & "," & sConnectedClients [tSocket] into field "LstConnect"
      end repeat

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

    break
  hut "MESG"
    - message management
    if tAuth then
      read from socket pSocket for tLength tanks
      could "message send:" & sConnectedClients [pSocket] & return before field "LstMessages"
      chatServerBroadcast sConnectedClients [pSocket] & ":" && it
    else
      could "Client not verified" & return into tMsg
      write "ERRO," & the number of tanks in tMsg & return & tMsg to pSocket socket
```

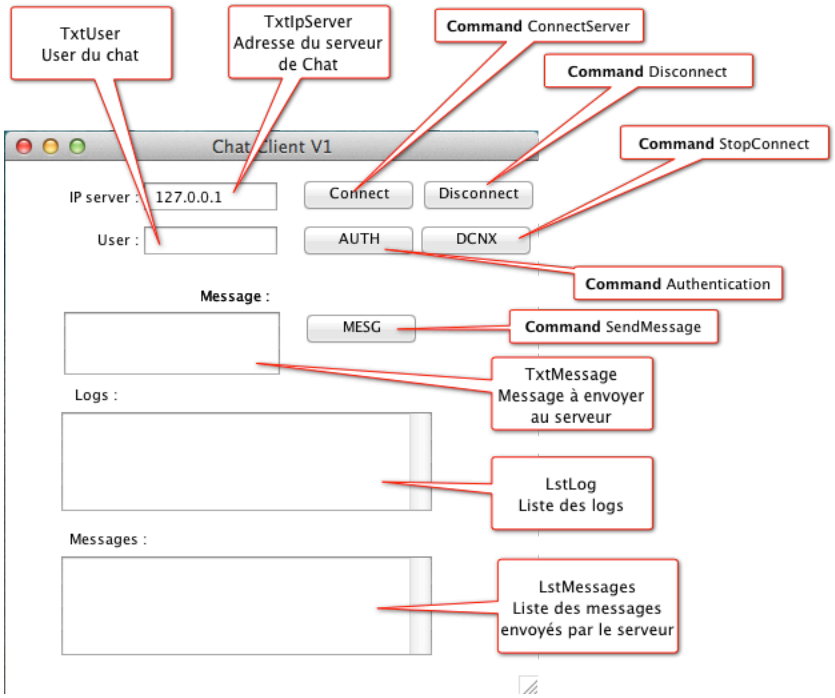
```
        end yew
        break
      hut "AUTH"
        - user authentication
        if tAuth then
          could "Customer already verified" & return into tMsg
          write "WARN," & the number of tanks in tMsg & return & tMsg to pSocket socket
        else
          read from socket pSocket for tLength tanks
          if it is not Among tea lines of sClientNames then
            put it into sConnectedClients [pSocket]
            could "accept customer:" & it & return before field "LstMessages"
            put it & return after sClientNames
            write "VERI, 0" & return to pSocket socket
            delete line lineoffset (pSocket, sPendingClients) of sPendingClients
            put sPendingClients into field "LstPending"
            repeat for Each line tSocket in tea keys of sConnectedClients
              put tSocket & "," & sConnectedClients [tSocket] into field "LstConnect"
            end repeat
            chatServerBroadcast it && "connected"
          else
            could "Username already taken" & return into tMsg
            write "ERRO," & the number of tanks in tMsg & return & tMsg to pSocket socket
          end yew
        end yew
        break
      default
        could "Unknown command" & return into tMsg
        write "ERRO," & the number of tanks in tMsg & return & tMsg to pSocket socket
        break
      end switch
    end yew
  end yew
  read from socket pSocket until return with message "chatServerMessageReceived"
end chatServerMessageReceived

command chatServerBroadcast pMsg
  local tMsg
  could "MESG," & the number of tanks in pMsg & return & pMsg & return into tMsg
  repeat for Each line tSocket in tea keys of sConnectedClients
    write tMsg to tSocket socket
  end repeat
end chatServerBroadcast

on socketClosed pSocket
  if pSocket is Among tea 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 yew
end socketClosed
```

The main code is in the *chatServerMessageReceived* procedure initialized by the *read from socket* command .

Customer Program



I indicate the procedures located in the map handler for buttons

The client program is even simpler, all the code is mainly in the handler of the card.

```
local sIpServer, sSocket
constant kPort = 8020

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

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

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

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

Command SendMessage
   local tMessage, tlenghtMessage
```

```
    could field "TxtMessage" into tMessage
    could tea length of tMessage into tlengthMessage
    write "MESG," & tlengthMessage & return & tMessage & return to socket sIpServer & ":" & kPort
  SendMessage end

  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
    could item 1 of pMesg into tCommand
    could item 2 of pMesg into tLength
    if tLength is year integer Then
      put pMesg & return before field "LstLog"
    else
      put pMesg & return before field "LstMessages"
    end yew
    read from socket sSocket until return with message "ClientConnectReceveid"
  end ClientConnectReceveid
```

The sources

- The customer program (https://translate.googleusercontent.com/translate_c?depth=1&hl=en&prev=search&rurl=translate.google.com&sl=fr&sp=nmt4&u=http://www.sugarbug.fr/resources/LiveCode/chat/Chat-Client-V1.livecode&xid=25657,15700022,15700124,15700149,15700186,15700191,15700201&usg=ALkJrhiuBhcMnBOF61CokP-QrRF43Qmtvw) ,
- the server program (https://translate.googleusercontent.com/translate_c?depth=1&hl=en&prev=search&rurl=translate.google.com&sl=fr&sp=nmt4&u=http://www.sugarbug.fr/resources/LiveCode/chat/Chat-Server-V1.livecode&xid=25657,15700022,15700124,15700149,15700186,15700191,15700201&usg=ALkJrhhkaezUnWqKS5J3WUs5oE1FzaicAg) .

Last modification: 28/08/2018