

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

The Kermith workshop (<https://translate.googleusercontent.com/depth=1&hl=en&prev=search&rurl=translate.google.com&sl>)

The other way to see supervision ...

Network sockets with LiveCode



Sockets allow communication between various processes using the TCP / IP layer. The communication can be realized in the machine itself with the local loop of the network interface or through the physical network between several machines. Historically, sockets have been implemented in Berkeley's UNIX distributions, sometimes referred to as Berkeley Software Distribution (BSD)

sockets.

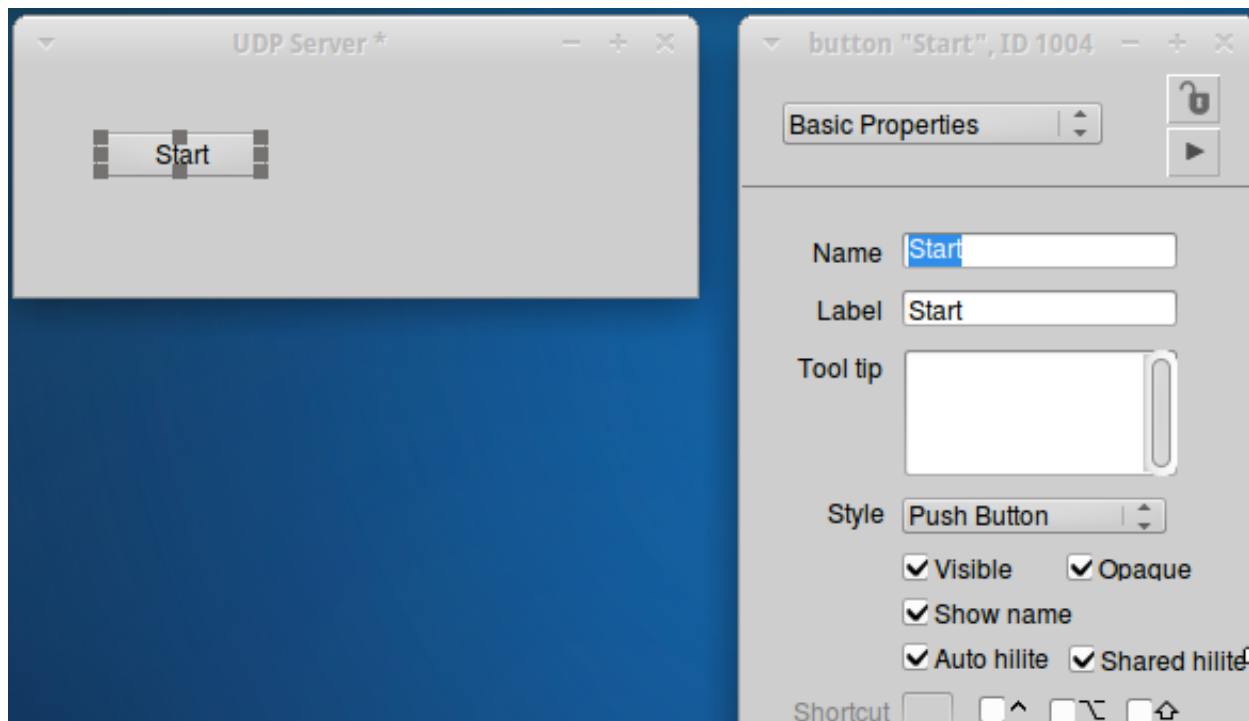
There are two types of communication:

Connected mode using the TCP protocol. This connection mode is comparable to a telephone call. A durable connection is established between the transmitter (client) and the receiver (server). The destination address is only needed when initializing the connection.

The unconnected mode using the UDP protocol. This connection mode is similar to sending a mail and does not require any acknowledgment of receipt. The destination address is required for each connection.

Creating our first UDP server

To realize our UDP server, we will use a virtual machine under Linux. This one will have for IP address 172.16.209.176. We will need a battery containing a button named "Start" and that's it. Name your UDP Server stack right away.



The UDP Server interface

Right-click on the button to get the Object Inspector. Enter Start in Name and Label. Then right-click the button again to open a code editor window. First, we will enter the code associated with the Start button.

on mouseUp

- **we check the label of the button**
- **start: starting the server**
- **stop: stop the server**

yew tea label of me = "Start" **Then**

- **we accept UDP connections on port 9997**
- **the dataReceived procedure is initialized**

accept datagram connections **we** Harbor 9997 with message "dataReceived"

- **we change the label of the button to stop**
- set** tea label of me to "Stop"

else

- **the UDP connection is stopped**
- close** socket 9997
- **we modify the label of the button to start**
- set** tea label of me to "Start"

end yew

end mouseUp

Explanation: Incoming UDP connections are accepted on port 9997. For each successful connection, the dataReceived (Message Handler) procedure will be executed.

You have to create the rest of the code.

```
on dataReceived pSocket, pMsg
    - display UDP message dialog box received
    answer pSocket & cr & pMsg
end dataReceived
```

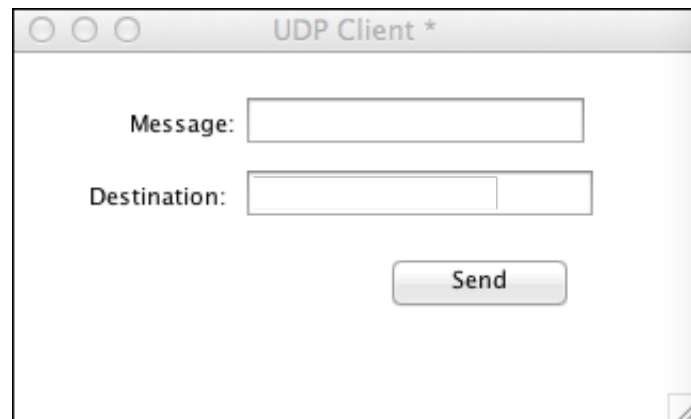
Explanation: The server receives the UDP frame and processes it with this procedure. We retrieve the address of the remote machine and the message.

```
on socketError
    - display UDP stream error dialog box
    answer "Failed to open server socket"
end socketError
```

Explanation: The socketError procedure is enabled when a problem occurs while establishing the UDP connection.

Creating the UDP client program

The client program will be realized in a Mac environment. We need a stack containing two text fields "txtMessage" and "ipDestination" and a button named "Send". Name your UDP client project.



Customer Socket Battery

Clicking the Send button will initialize the TCP connection. Here is the code.

```
on mouseUp
    - verification of a message
    yew eld "txtMessage" is not empty Then
        local ipDest
        - initialization of the ipDest variable
        - ipDest destination address of the UDP server accompanied by the port number
        could eld "ipDestination" into ipDest
        could ": 9997" after ipDest
```

```

- opens a UDP stream
open datagram socket to ipDest
yew tea result is not empty Then
  - pb network we send the result
  answer "Socket:" & the result
end yew
- the message is sent without acknowledgment of receipt
write eld "txtMessage" to ipDest socket
- we close the UDP connection
close ipDest socket
- we delete the message
could empty into eld "txtMessage"
else
  beep
end yew
end mouseUp

```

Explanation: we verify the existence of a message in the text field, then we create the UDP connection and send the message directly to port 9997. we closed the communication because we do not wait for acknowledgment of receipt server. At the end, we erase the message.

```

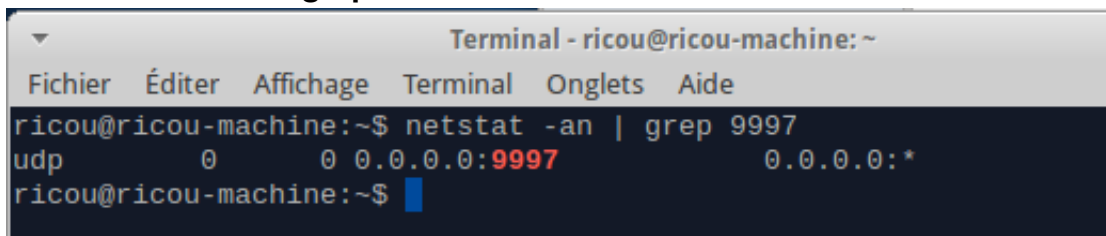
on socketError
  answer "Failed to open socket"
end socketError

```

As before, this procedure intercepts TCP stream errors

Operation

We must start our UDP server on Linux by clicking the Start button. To check that it works, use the command **netstat -an | grep 9997** on the command line.



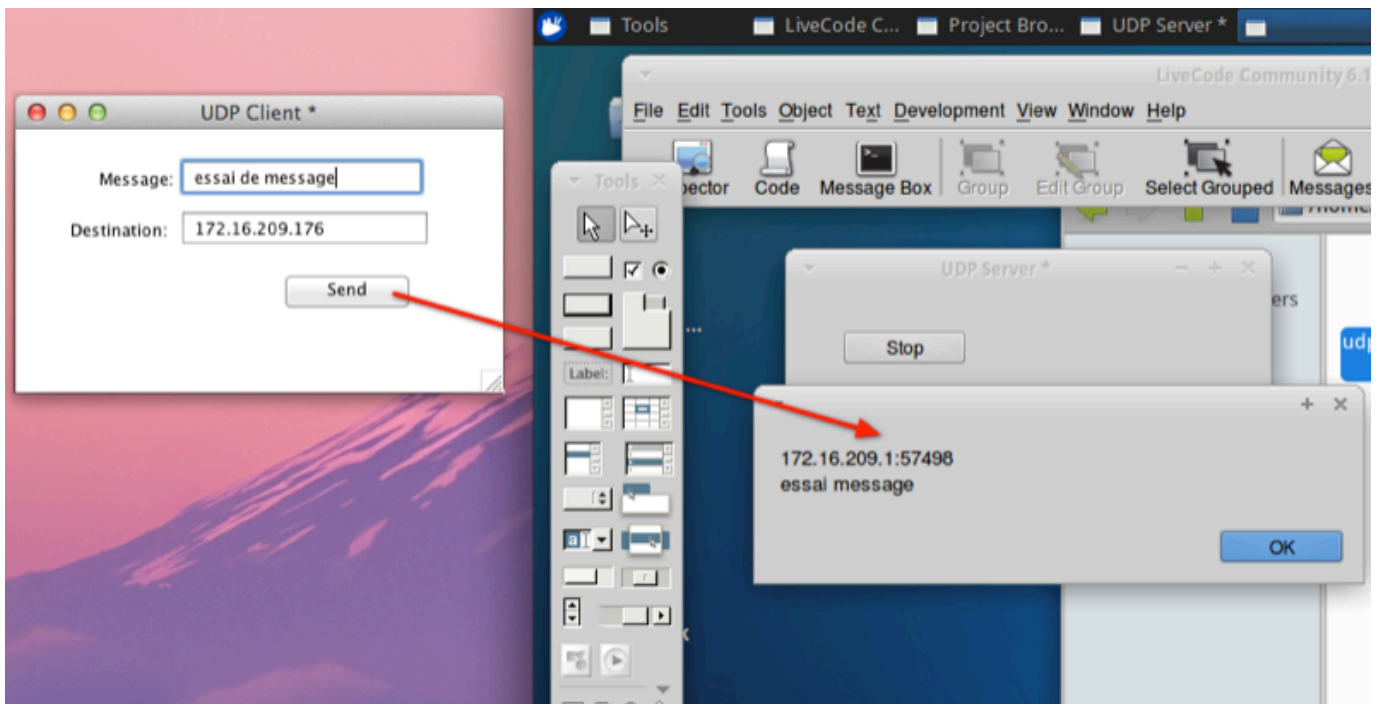
```

Terminal - ricou@ricou-machine: ~
Fichier Éditer Affichage Terminal Onglets Aide
ricou@ricou-machine:~$ netstat -an | grep 9997
udp        0      0 0.0.0.0:9997         0.0.0.0:*
ricou@ricou-machine:~$

```

UDP port 9997 is listening.

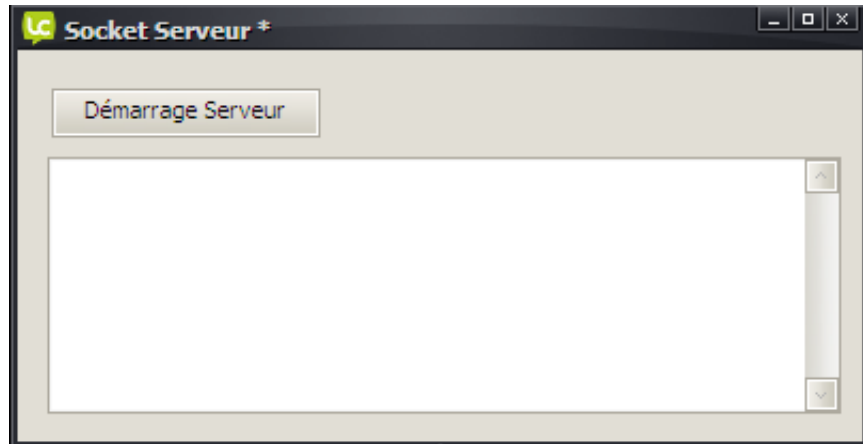
Then, enter a message in the client UDP program and click the send button. A dialog box should appear on the Linux computer.



How the UDP connection works

Creating our first TCP server

Once is not customary, we will use a machine under Windows to build our first server. This one will have for IP address 172.16.209.176. We will need a stack containing a button and a list called LstMessage. Name our stack Socket Server.



Map of our server

Right-click the button to open a code editor window. First, we will enter the code associated with the Start Server button.

```
on mouseUp
  accept connections we Harbor 20001 with message "MachineConnecte"
end mouseUp
```

Explanation: Incoming TCP connections are accepted on port 20001. For each successful connection, the MachineConnect (Message Handler) procedure will be executed.

You have to create the rest of the code.

```
on MachineConnection ipMachine
```

```
  read from ipMachine until socket return with message "new message"
```

```
End MachineConnect
```

Explanation: The connection is established, we initialize the procedure newMessage. The TCP stream will use this procedure as long as the connection is active.

```
newMessage ipMachine theMessage
```

```
put ipMachine && ":" && theMessage & return after- field "LstMessage"
```

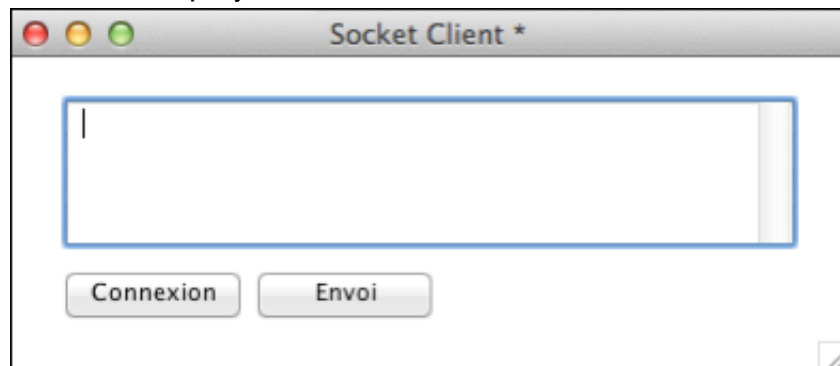
```
  read from ipMachine until socket return with message "new message"
```

```
end newMessage
```

Explanation: The client data is retrieved by the newMessage procedure. We retrieve the IP address and the message in the LstMessage list. Each message is delimited by a carriage return. The last line is used to process the following messages.

Creating our TCP client program

The client program will be realized in a Mac environment. We need a stack containing two buttons and a list. Name our project Socket client.



Customer Socket Battery

Connection button

This button will be used to initialize the TCP connection. Here is the code.

```
on mouseUp
```

```
  open socket to "172.16.209.176:20001"
```

```
    yew tea Result <> "" Then
```

```
      could "result:" && the result
```

```
    end yew
```

```
end mouseUp
```

Explanation: The open socket command attempts to open a TCP connection with port 20001 and server address 172.16.209.176. If the connection is already active, the result is displayed in the MessageBox window.

Send button

This button will be used to send the message stored in the LstMessage list. Here is the code.

```
on mouseUp
    write field "LstMessage" & return to socket "172.16.209.176:20001"
end mouseUp
```

Explanation: The LstMessage content is sent to the 172.16.209.176 server with the 20001 port.

Operation

We must start our server by clicking the Start Server button. To verify that it works, use the netstat -an command line command.

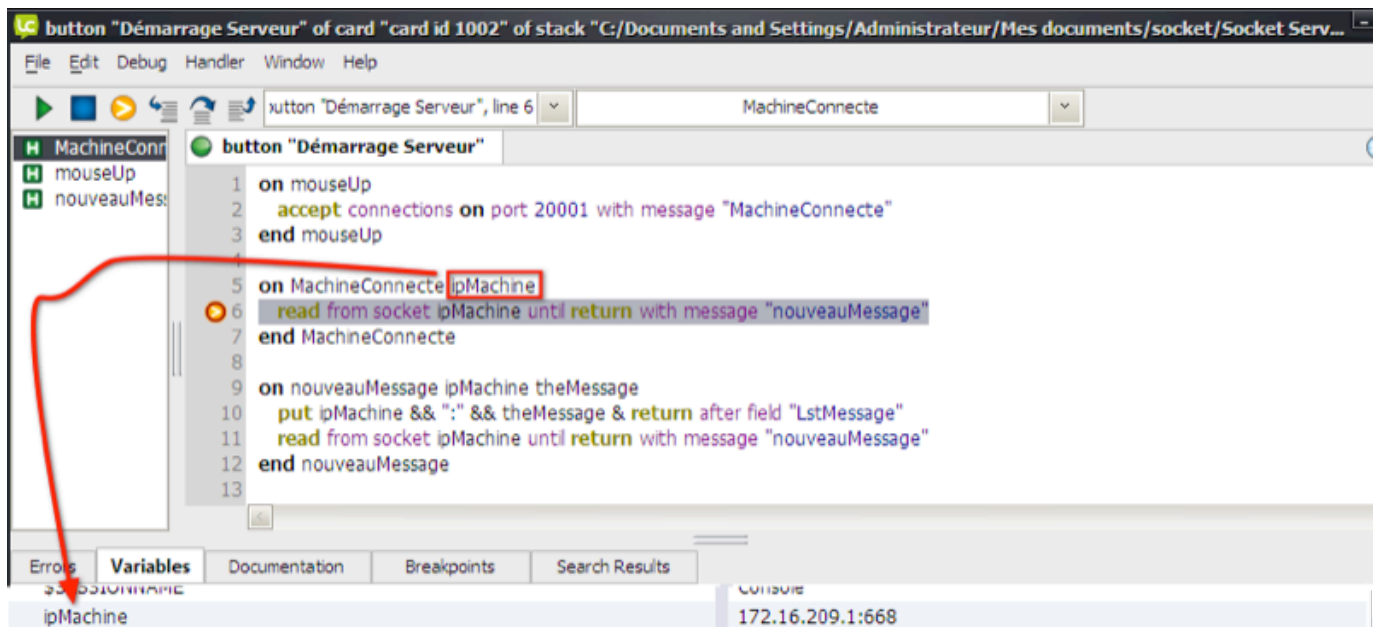
```
C:\Documents and Settings\Administrateur>netstat -an

Connexions actives

Proto  Adresse locale      Adresse distante     État
TCP    0.0.0.0:990          0.0.0.0:0            En écoute
TCP    0.0.0.0:20001        0.0.0.0:0            En écoute
TCP    127.0.0.1:5679       0.0.0.0:0            En écoute
TCP    127.0.0.1:7438       0.0.0.0:0            En écoute
TCP    172.16.209.176:139   0.0.0.0:0            En écoute
UDP    172.16.209.176:137   *:*
```

TCP 20001 port is listening (LISTEN)

Then click on the client connection button. A breakpoint in the MachineConnection procedure tells us the establishment of the connection.



Establishing the TCP connection

We can check the TCP connection on the server side:

```
> netstat -an | grep 20001
```

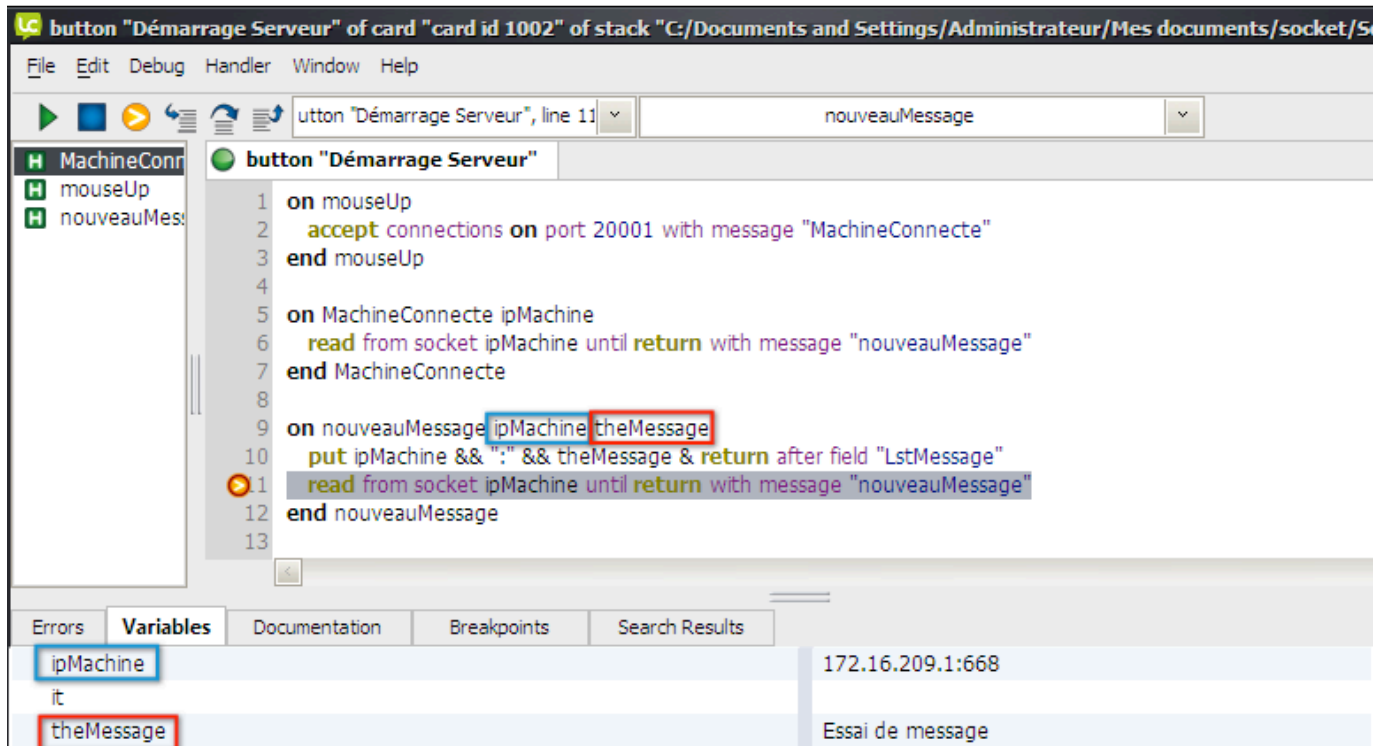
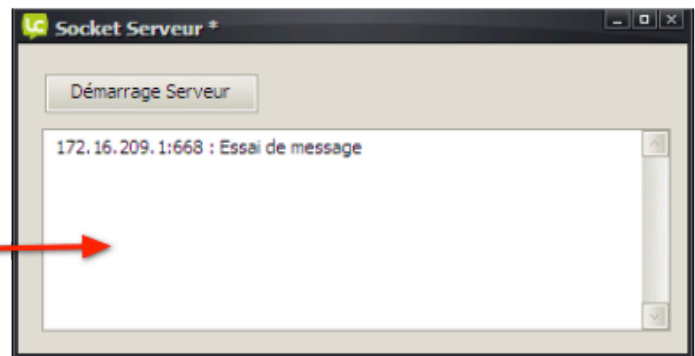
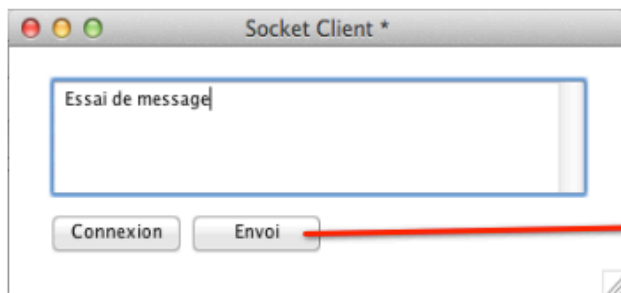
```
TCP 172.16.209.176:20001 172.16.209.1:51545 Established
```

and client side:

```
> netstat -an | grep 20001
```

```
tcp4 0 0 172.16.209.1.51545 172.16.209.176.20001 ESTABLISHED
```

Let's continue by sending a text to the server. On the client side, we enter "message test" and click on the send button.



How the TCP stream works

You've seen a brief overview of what can be done with Livecode and BSD sockets. Feel free to improve these small programs. Your keyboards !

I added an article on the encryption of UDP data that you will find [here](#) ([../code/LiveCodeSocket/LiveCodeSocket_Secure/](#)).

comments powered by Disqus (<http://disqus.com>).