```
import select
import socket
import sys

debugPrint = 1
sent = 0
onthefly = 0
INIT = 888
READY = 4
ALL_CLIENTS = 4444
clientSize = 15
clientIndex = []

for i in range (clientSize) : clientIndex.append(0)
clientAction = []
for i in range (clientSize) : clientAction.append(0)

# initialize server sockets
host = ''
port = 51318
backlog = 15
size = 1024
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((host,port))
server.listen(backlog)
input = [server]

def CheckServerConnections ( ) :
    global clientAction
    global clientIndex
    global clientSize
    global INIT
    global READY
    global ALL_CLIENTS
    global input
    global server

    inputready,outputready,exceptready = select.select(input,[],[], .1)  # .1 second
timeout (10 times per second)

    for s in inputready:  # loop over all ready sockets, none if timeout

        if s == server:
            # handle the server socket
            print 'server connected'
            client, address = server.accept()
            input.append(client)
            for i in range (clientSize) :  # add socket into socket index
```

```
                if clientAction[i] == 0 :
                    clientAction[i] = INIT
                    clientIndex[i] = client
                    if debugPrint > 0 : print 'adding client at index',i, client,
address
                    break
        else:
            # handle all other sockets
            try :
                data = s.recv(size)  # get data from socket
            except :
                # error getting data, remove socket from index array and close
socket
                if debugPrint > 0 : print 'exception in recv - closing socket'
                i = FindSocketIndex (s)
                clientAction[i] = 0
                clientIndex[i] = 0
                s.close();
                input.remove(s)
                continue
            if data :
                index = FindSocketIndex (s)
                if debugPrint > 0 : print "data received", data
                ReceivedData (index, data)
            else:
                print "closing"
                i = FindSocketIndex (s)
                clientAction[i] = 0
                clientIndex[i] = 0
                s.close()
                input.remove(s)
    return
#end CheckSocketConnections

def FindSocketIndex (socket) :
    global clientAction
    global clientIndex
    global clientSize

    for i in range (clientSize) :
        print 'looking for client',socket, clientIndex[i], clientAction[i]
        if clientIndex[i] == socket :
            break
    return i

#end FindSocketIndex
```

```
def sendToSocket (indexArray, string) :
    global clientIndex
    global clientAction
    global clientSize
    global ALL_CLIENTS
    global INIT
    global READY

    if indexArray == ALL_CLIENTS :
        for i in range (clientSize) :
            if clientAction[i] == READY :
                try :
                    if debugPrint > 0 : print "index", indexArray, "sock send",
string

                    clientIndex[i].send (string)
                except :
                    if debugPrint > 0 : print "error sending to socket index", i,
string
                    continue
    else :
        try:
            clientIndex[indexArray].send (string)
        except :
            if debugPrint > 0 : print "error sending to socket sindex", indexArray,
string

    return
#end sendToClient

def ReceivedData (indexArray, string) :
    global clientIndex
    global clientAction
    global clientSize
    global ALL_CLIENTS
    global INIT
    global READY

    if debugPrint > 0 : print "received index:", indexArray, 'array state',
clientAction[indexArray], 'data', string

    if clientAction[indexArray] == INIT :
        PopulateClient (indexArray)
        clientAction[indexArray] = READY
    else :
        ParseReceivedData (indexArray, string)
    return
#end ReceivedData
```

```
def ParseReceivedData (outSocket, receivedData) :
    global sent
    global onthefly
    global colorOn
    global nextColor
    global ALL_CLIENTS

    sent += 1

    button_name = receivedData[1: 1 + receivedData[1:].index("^")]

    if receivedData.find("DISPLAY$FLY") != -1 :

        DisplayBuildOnTheFly (outSocket)
        onthefly = 1
        colorOn = 0  # turn off color updates if they are on


    if button_name[0:5] == "COLOR" :
       colorOn = (colorOn + 1) % 2
       nextColor = 0

    else :
        colorOn = 0  # turn off color updates if they are on

        button_added_length = receivedData[ receivedData[:-3].rindex("^") + 1 :
len(receivedData) - 3]
        if debugPrint > 0 : print 'button added len',button_added_length
        # if debugPrint > 0 : print "added",button_added_length, "len",
len(button_added_length)

        if button_added_length == "0" :
            out = "^"+ button_name + "^TEXT^\nHELLO^\r\n"
        else :
            out = "^"+ button_name + "^TEXT^^\r\n"
        if sent % 2 == 0 :
            out = "^"+ button_name + "^ON^\r\n"
        if sent % 3 == 0 :
            out = "^"+ button_name + "^OFF^\r\n"

        sendToSocket (ALL_CLIENTS, out)

#end ParseReceivedData


def PopulateClient (indexArray) :
```

```
    # build TTT row of buttons.

    #out = "^INI+TTT1^1:1^TTT 1^TOP^TTT1^^\r\n"
    #sendToSocket (indexArray, out)
    #out = "^INI+TTT2^2:1^TTT 2 ignore press^TOP^IGNORE^^\r\n"
    #sendToSocket (indexArray,out)

    DisplayToolBar(indexArray)
    DisplayKitchenLights(indexArray)
    DisplayUpstairs(indexArray)
    DisplayColRow (indexArray)
    DisplayBoxes (indexArray)
    DisplayTTTForBuildOnFly (indexArray)
    DisplayColors (indexArray)

    out3 = "^TTT1^DISPLAY^^\r\n"  # force display of TTT device
    #sendToSocket (indexArray, out3)

#end PopulateClient
def DisplayToolBar(out_socketIndex):


    sendToSocket( out_socketIndex, "^INI+TTT1^1:1^Temperature^TOP^IGNORE^^\r\n")
    sendToSocket
(out_socketIndex,"^INI+TTT2^2:1^UpStairs^TOP^DISPLAY$Upstairs^^\r\n")
    sendToSocket
(out_socketIndex,"^INI+KitchenTop^3:1^Kitchen^TOP^DISPLAY$Kitchen^^\r\n")
    sendToSocket (out_socketIndex, "^INI+TTTColRow^4:1^Display Col
Row^TOP^DISPLAY$COLROW^^\r\n")
    sendToSocket (out_socketIndex, "^INI+TTTBoxes^5:1^Popup Box
Buttons^TOP^DISPLAY$BOXES^^\r\n")
    sendToSocket (out_socketIndex, "^INI+TTT3^6:1^Build on the
fly^TOP^DISPLAY$BUILD^^\r\n" )

    sendToSocket (out_socketIndex,  "^INI+TTTcolor^7:1^Display
Colors^TOP^DISPLAY$COLOR^^\r\n")



def DisplayKitchenLights(outSocketIndex) :

    sendToSocket( outSocketIndex, "^INI+KIT1^1:2^Main
Lights^Kitchen^Kit1^^PINK^\r\n")
    sendToSocket( outSocketIndex, "^INI+KIT2^2:2^Nook^Kitchen^Kit2.2^^ORANGE^\r\n")
    sendToSocket( outSocketIndex, "^INI+KIT3^3:2^Night
Lights^Kitchen^Kit3.3^^GREEN^\r\n")
    sendToSocket( outSocketIndex, "^INI+KIT4^4:2^Accent
Lights^Kitchen^Kit4.4^^SILVER^\r\n")
```

```
    sendToSocket( outSocketIndex, "^INI+KIT5^1:3^Pantry
Lights^Kitchen^Kit5.5^^indigo^\r\n")
    sendToSocket( outSocketIndex, "^INI+KIT6^2:3^Hall
Lights^Kitchen^Kit6.6^^red^\r\n")
    sendToSocket( outSocketIndex, "^INI+KIT7^3:3^Outside
Lights^Kitchen^Kit7.7^^red^\r\n")
    sendToSocket( outSocketIndex, "^INI+KIT8^4:3^Counter^Kitchen^Kit8.8^^pink^\r\n")
    sendToSocket( outSocketIndex, "^INI+KIT9^1:4^Evening
Lights^Kitchen^Kit9.8^^grey^\r\n")
    sendToSocket( outSocketIndex, "^INI+KIT10^2:4^Dining
Room^Kitchen^Kit9.9^^goldenrod^\r\n")
    sendToSocket( outSocketIndex, "^INI+KIT11^3:5^All
Off^Kitchen^KitAllOff^^purple^\r\n")

def DisplayUpstairs(outSocketIndex) :

    sendToSocket( outSocketIndex, "^INI+Upstairs1^1:2^Hallway^Upstairs^Up1^^\r\n")
    sendToSocket( outSocketIndex, "^INI+Upstairs2^2:2^Bedroom
1^Upstairs^Up2.2^^\r\n")
    sendToSocket( outSocketIndex, "^INI+Upstairs3^3:2^Night
Lights^Upstairs^Up3.3^^\r\n")
    sendToSocket( outSocketIndex, "^INI+Upstairs4^4:2^Bedroom
2^Upstairs^Up4.4^^\r\n")
    sendToSocket( outSocketIndex, "^INI+Upstairs5^1:3^Bedroom
3^Upstairs^Up5.5^^\r\n")
    sendToSocket( outSocketIndex,
"^INI+Upstairs6^2:3^Stairwell^Upstairs^Up6.6^^\r\n")
    sendToSocket( outSocketIndex, "^INI+Upstairs7^3:3^Bathroom
1^Upstairs^Up7.7^^\r\n")
    sendToSocket( outSocketIndex, "^INI+Upstairs8^4:3^Bathroom
2^Upstairs^Up8.8^^\r\n")
    sendToSocket( outSocketIndex, "^INI+Upstairs11^3:5^All
Off^Upstairs^UpAllOff^^\r\n")

def DisplayColRow (out_socket) :

    sendToSocket (out_socket,
"^INI+XXXColRow^1:2^BACK^COLROW^DISPLAY$Kitchen^^\r\n")

    for i in range (6) :
          c1 = i + 2
          r1 = i + 2
          out3 = "^INI+COLROW" + str(c1) + str(r1) + "^" + str(c1) + ":" + str(r1) +
"^Col "+ str(c1) + "\n Row " + str(r1) + "^COLROW^rowcol^^\r\n"
          sendToSocket (out_socket, out3)
          r1 = 9 - r1
          out3 = "^INI+COLROW" + str(c1) + str(r1) + "^" + str(c1) + ":" + str(r1) +
"^Col "+ str(c1) + "\n Row " + str(r1) + "^COLROW^rowcol^^\r\n"
```

```
          sendToSocket (out_socket, out3)
    return

#end DisplayColRow

def DisplayBoxes (out_socket) :

    sendToSocket (out_socket,  "^INI+XXXboxes^1:2^BACK^BOXES^DISPLAY$Kitchen^^\r\n")

    for i in range (6) :
          c1 = i + 2
          r1 = i + 2
          out3 = "^INI+BOXES" + str(c1) + str(r1) + "^" + str(c1) + ":" + str(r1) +
"^Box "+ str(c1) + "\n Row " + str(r1) + "^BOXES^rowcol^BOX^\r\n"
          sendToSocket (out_socket, out3)
          r1 = 9 - r1
          out3 = "^INI+boxes" + str(c1) + str(r1) + "^" + str(c1) + ":" + str(r1) +
"^Box "+ str(c1) + "\n Row " + str(r1) + "^BOXES^rowcol^BOX^\r\n"
          sendToSocket (out_socket, out3)
    return

#end DisplayBoxes

def DisplayTTTForBuildOnFly (out_socket) :

     sendToSocket (out_socket,
"^INI+XXXbuild^1:2^BACK^BUILD^DISPLAY$Kitchen^^\r\n")
     out = "^INI+Build^0^build 1^BUILD^DISPLAY$FLY^^\r\n"
     sendToSocket (out_socket, out)
     return

def DisplayBuildOnTheFly (out_socket) :

    sendToSocket (out_socket,  "^INI+XXXbfly^1:1^BACK^FLY^DISPLAY$Kitchen^^\r\n")
    for i in range (6) :
          c1 = i + 2
          r1 = i + 2
          out3 = "^INI+FLY" + str(c1) + str(r1) + "^" + str(c1) + ":" + str(r1) +
"^Fly "+ str(c1) + "\n Row " + str(r1) + "^FLY^rowcol^^\r\n"

          sendToSocket (out_socket, out3)
          r1 = 9 - r1
          out3 = "^INI+fly" + str(c1) + str(r1) + "^" + str(c1) + ":" + str(r1) +
"^Fly "+ str(c1) + "\n Row " + str(r1) + "^FLY^rowcol^^\r\n"
          sendToSocket (out_socket, out3)
    out3 = "^FLY22^DISPLAY^\r\n"
    sendToSocket (out_socket, out3)
    return
```

```
#end DisplayBuildOnTheFly


def DisplayColors (out_socket) :

    for i in range (8) :
       for j in range (8) :
          c1 = i + 1
          r1 = j + 1
          out3 = "^INI+COLOR" + str(c1) + str(r1) + "^" + str(c1) + ":" + str(r1) +
"^Col "+ str(c1) + "\n Row " + str(r1) + "^COLOR^rowcol^^\r\n"
          if c1 == 1 and r1 == 1 :
              out3 = "^INI+COLOR" + str(c1) + str(r1) + "^" + str(c1) + ":" +
str(r1) + "^Col "+ str(c1) + "\n Row " + str(r1) + "^COLOR^DISPLAY$TTT^^\r\n"
          sendToSocket (out_socket, out3)

    return
#end DisplayColRow

colorOn = 0
nextColor = 0
colorCount = 0
colorChoice =
['Red','Green','Blue','Pink',"Cyan","Coral","Brown","Gold","Plum","MediumSpringGreen
","LightGoldenRodYellow", "Purple"]

def UpdateColors (index, count) :
    global colorOn
    global nextColor
    global colorCount;

    if count % 2 == 0 :
        if colorOn == 1 :

            tColor = nextColor % 63
            r1 = tColor % 8 + 1
            c1 = tColor / 8 + 1
            nextColor += 1
            iColor = nextColor % len(colorChoice)
            out3 = "^COLOR" + str(c1) + str(r1) + "^COLOR^" + colorChoice[iColor]+
"^^\r\n"
            if debugPrint > 0 : print "sending:", out3
            sendToSocket (index, out3)
            out3 = "^COLOR" + str(c1) + str(r1) + "^NEW^" + colorChoice[iColor]+
"^^\r\n"
            sendToSocket (index, out3)

            colorCount = colorCount + 1;
```

```
            if colorCount % 100 == 0 :
               colorOn = 0
               colorCount = 0


#end UpdateColors

def AsyncUpdate (index, count) :

    # every 2 sec do something
    if count % 20 != 0 :
        return

    atLeastOneClient = 0
    for i in range (clientSize) :
         if clientAction[i] == READY :
             atLeastOneClient = 1
             break

    if atLeastOneClient :
        out1 = "^TTT1^TEXT^\n" + str(count) + "^\r\n"
        out2 = "^TTT1^COLOR^Red^\r\n"
        if count % 100 : out2 = "^TTT1^COLOR^Green^\r\n"
        if count % 200 : out2 = "^TTT1^COLOR^Blue^\r\n"
        sendToSocket (index, out1)
        sendToSocket (index, out2)
    return

#end AsyncUpdate

running = 1
count = 0
while running:

    CheckServerConnections ()

    count += 1
    AsyncUpdate (ALL_CLIENTS, count)
    UpdateColors (ALL_CLIENTS, count)


server.close()
```