24.1.2.1 Exkurs – Sockets

Die in diesem Kapitel beschriebenen Netzwerk-Aspekte beziehen sich auf *Localhost* und *Sockets*. Es wird versucht zu zeigen, wie Socket-Theorie in Eigenschaften, Methoden und Ereignissen der Gambas-Klassen Socket und Server-Socket abgebildet wird und wie diese Klassen in Server- sowie Client-Programmen eingesetzt werden.

Dieser Exkurs soll das grundlegende Verständnis für den Einsatz von Sockets als Schnittstelle zur Programmierung von Netzwerkanwendungen mit Gambas fördern.

24.1.2.1.1 Localhost

Localhost ist nicht nur die Bezeichnung für den ihn repräsentierenden virtuellen Server auf einem Rechner, sondern auch dessen Domain-Name. Localhost hat die von der ICANN fest zugewiesene IPv4-Adresse 127.0.0.1, die auf den Server auf dem eigenen Rechner zeigt. Die entsprechende IPv6-Adresse ist mit ::1 reserviert.

Unter https://www.de.paessler.com/it-explained/ip-address finden Sie eine kurze Beschreibung zum Thema IP-Adressen sowie weitere interessante Aspekte bei diesen Webseiten:

- https://www.hosttest.de/artikel/was-ist-der-localhost-1440.html
- https://www.ionos.de/digitalguide/server/knowhow/localhost/
- https://www.onlinesolutionsgroup.de/blog/glossar/l/localhost/

Wenn Sie versuchen, die Domain in einem Browser mit http://127.0.0.1 oder http://localhost aufzurufen, so wird eine Loopback-Schleife ausgelöst. Die Anfrage wird nicht über den Router ins Internet weitergeleitet. Damit der Verweis auf den eigenen Rechner funktioniert, wird vom Betriebssystem das so genannte Loopback-Device erzeugt. Mit dem Befehl *ifconfig* in einer Konsole erhalten Sie wesentliche Informationen zur entsprechenden virtuellen Netz-Schnittstelle lo:

```
hans@mint-183 ~ $ ifconfig
lo        Link encap:Lokale Schleife
          inet Adresse:127.0.0.1  Maske:255.0.0.0
          inet6-Adresse: ::1/128 Gültigkeitsbereich:Maschine
          UP LOOPBACK RUNNING  MTU:65536  Metrik:1
          RX-Pakete:3716 Fehler:0 Verloren:0 Überläufe:0 Fenster:0
          TX-Pakete:3716 Fehler:0 Verloren:0 Überläufe:0 Träger:0
          Kollisionen:0 Sendewarteschlangenlänge:1000
          RX-Bytes:1393737 (1.3 MB)  TX-Bytes:1393737 (1.3 MB)
```

Mit dem Ping-Befehl – hier mit drei Pings – können Sie feststellen, ob und mit welchen Laufzeiten der Localhost auf Ihrem Rechner erreichbar ist:

```
hans@mint-183 ~ $ ping localhost -c 3
PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.026 ms
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.038 ms
64 bytes from localhost (127.0.0.1): icmp_seq=3 ttl=64 time=0.038 ms
--- localhost ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2032ms
rtt min/avg/max/mdev = 0.026/0.034/0.038/0.005 ms
```

Durch die Möglichkeit, den lokalen Rechner in Netzwerkanwendungen immer als 'localhost' zu adressieren, ist die Herstellung eine TCP-Verbindung auch ohne Kenntnis des Hostnamens des Rechners oder der genauen IP-Adresse möglich.

Die virtuelle Netz-Schnittstelle reagiert auf jede lokale IPv4-Adresse im Adressbereich 127.0.0.1/8. Sie leitet anschließend eine Verbindung von 127.0.0.1 bis 127.255.255.254 beziehungsweise der Domain localhost auf sich selbst um. Wenn eine IP-Adresse mit dem Block 127. aus dem Bereich 127.0.0.1 bis 127.255.255.254 beginnt, so werden diese IP-Adressen mit dem Domain-Namen localhost identifiziert und auf 127.0.0.1 gesetzt! Jede IP-Adresse, die mit 127. beginnt, wird als Loopback-Adresse betrachtet.

24.1.2.1.2 Socket

In ersten Teil wird ein konzeptioneller Überblick zur Socket-Programmierung für TCP/IP-Server-Client-

Anwendungen gegeben. Erst im zweiten Teil wird auf Details der Gambas-Programmierung eingegangen.

Eine elementare Schnittstelle zur Programmierung von Netzwerkanwendungen bieten die so genannten Sockets, denn sie stellen Funktionen zum Aufbau von Netzwerk-Verbindungen sowie zur Kommunikation zwischen Anwendungen in Netzwerken bereit.

- Sockets sind Netzwerkverbindungsendpunkte.
- Sockets werden von allen gängigen Betriebssystemen unterstützt.
- Sockets kapseln die Details der Netzwerkkommunikation.
- Sockets unterstützen *verbindungsorientierte* und *verbindungslose* Netzwerkkommunikation.
- Sockets setzen auf Protokolle wie TCP oder UDP auf. Der Datenaustausch erfolgt über entweder über Streams (TCP) oder über Datagramme (UDP).
- Sockets benötigen für die Netzwerkkommunikation stets eine vollständige Adresse, die aus IP-Adresse und Port-Nummer (Port) besteht.
- Das Erzeugen und konfigurieren eines (Server-)Sockets erfolgt unabhängig vom Dienst, den ein Server anbietet.

Das Socket-Interface bietet einer Netzwerkanwendung die Möglichkeit über TCP (stream socket), UDP (datagram socket) oder direkt über eine Datei (Unix-Socket) die Verbindung zwischen Client und Server aufzubauen.

Netzwerkanwendungen werden als Server-Client-Anwendungen konzipiert, bei denen ein Server einen Dienst (Service) anbietet, der von vielen Clients genutzt werden kann. Aus diesem Grunde gibt es in Gambas (einfache) Sockets und Server-Sockets.

- Wenn Sie einen Service nutzen wollen, dann müssen Sie sowohl die IP-Adresse des Servers als auch den Port kennen! Die IP-Adresse und der Port sind feste Werte.
- Unterschiedliche Dienste, die auf einem Server im gleichen Sub-Netz und daher unter der selben IP-Adresse laufen, werden durch ihre (unterschiedliche) Port-Nummer identifiziert!
- Ein Client dagegen braucht beim Start keinen festen Port! Er fordert vom System dynamisch die nächste verfügbare, aktuell nicht verwendete Port-Nummer im Subnetz an, um den erzeugten Socket an diese Port-Nummer zu binden sowie diesen Port zu öffnen. Damit wird zusammen mit der IP-Adresse auch für den Client eine eindeutige Adressierung möglich.
- Der Server erfährt die vollständige Adresse des Clients aus der Anfrage des Clients und kann dann mit ihm unter dieser vollständigen Adresse kommunizieren.

Bei der Vergabe von Port-Nummern für eigene Netzwerkanwendungen (Server) sollten Sie Folgendes beachten:

- 1 - 1023: Dieser Bereich der Port-Nummern ist für Standard-Dienste (well known services) wie 80 für HTTP oder 21 für FTP vorgesehen.
- 1024 - 49151: Auch die Port-Nummern in diesem Bereich sind registriert, werden aber nicht überwacht.
- 49152 – 65535: Diese Port-Nummern werden vom System zum Beispiel für dynamisch erzeugte Sockets automatisch vergeben.

Beachten Sie: In der Socket-Programmierung hat Port 0 eine besondere Bedeutung, denn Port 0 ist ein reservierter Port in Netzwerken. Im Kapitel 24.1.3.1 UDPSocket-Projekte wird diese Besonderheit genauer beschrieben.

24.1.2.1.3 Server, Server-Socket und Kommunikation (Datenaustausch)

Die Rolle als Server in einer Netzwerkanwendung spiegelt sich in folgender Aufstellung wider, wobei die Socket-Systemaufrufe in Klammern angegeben werden:

1. ServerSocket von speziellem Server-Typ (TCP oder UDP) als Netzwerkverbindungsendpunkt erzeugen [socket()].
2. Vollständige Adresse festlegen: IP-Adresse im Sub-Netz übernehmen und an festlegten Port binden [bind()].
3. Warteschlange starten - Maximale Anzahl von Client-Anfragen festlegen [listen(max)].

4. Auf Verbindungsanfragen von Clients lauschen: Eingehende Client-Anfrage akzeptieren → Erzeugen eines Kommunikationssockets auf dem Server für die Kommunikation zwischen Server und akzeptiertem Client [accept()] oder die Client-Anfrage ablehnen.
5. Netzwerk-Kommunikation: Daten empfangen [read()] und Daten senden [write()] sowie Status überwachen.
6. Kommunikationssockets schließen, wenn ein Client die Verbindung zum Server schließt [close()].
7. Server-Socket schließen [close()].

24.1.2.1.4 Server-Projekt

Ein Modell-Server – der genau die o.a. sieben Punkte umsetzt – implementiert im Beispiel einen sehr einfachen Service oder Dienst. Der Service besteht darin, dass Daten, die ein mit dem Server verbundener Client an den Server sendet, nur mit einem 'ok'-Kommentar an den Client zurückgeschickt werden – der Server bietet den 'OK'-Dienst an.

Hinweis:
Auf der Basis des Modell-Servers werden Ihnen im nächsten Kapitel zwei Server-Client-Anwendungen und eine Client-Anwendung vorgestellt.

Der Gambas-Quelltext für einen Modell-Server ist umfangreich kommentiert:

```
[1]   ' Gambas class file
[2]
[3]   Public ServerSocket As ServerSocket
[4]   Public cConnetedSockets As Collection '-- Multi-Client-Betrieb
[5]   Public sIPAddress As String
[6]
[7]   Public Sub Form_Open()
[8]
[9]       Dim sHostname As String
[10]
[11]      Shell ("hostname -I | grep -oE '" & Chr$(40) & "[[:digit:]]{1,3}" & Chr$(92) & "." & Chr$(41) & \
[12]            "{3}[[:digit:]]{1,3}'") To sIPAddress
[13]      Shell ("hostname") To sHostname
[14]
[15]      FMain.Caption = "TCPServer | IP-Adresse: " & Trim(sIPAddress) & " | Hostname: " & Trim(sHostname)
[16]      cConnetedSockets = New Collection
[17]
[18]  '-- Es wird ein Server-Socket erzeugt
[19]      ServerSocket = New ServerSocket As "ServerSocket"
[20]  '-- Der Server-Typ wird auf `Internet Domain Socket` festgelegt
[21]      ServerSocket.Type = Net.Internet
[22]  '-- Der implementierte Dienst/Service wird an den vorgegebenen Port gebunden (bind())
[23]      ServerSocket.Port = Val(txbPort.Text)
[24]
[25]      btnServiceStartStop.Caption = "Service starten"
[26]      MAddOns.SetLEDColor(picBoxOnOff, "red")
[27]
[28]  End
[29]
[30]  Public Sub btnServiceStartStop_Click()
[31]
[32]      If ServerSocket.Status <= Net.Inactive Then
[33]     '-- Beginnt das Hören am ausgewählten TCP-Port. Optional können Sie auch einen einzelnen Parameter MaxConn übergeben:
[34]     '-- MaxConn kann 0 (keine Verbindungsbeschränkung) oder größer als 0 sein, was die maximale Anzahl der gleichzeitig
[35]     '-- aktiven Verbindungen angibt. Dieser Parameter ist optional.
[36]     '-- Wenn Sie ihn nicht verwenden oder auf 0 setzen, gibt es keine Beschränkung der Anzahl der Client-Verbindungen.
[37]         ServerSocket.Listen(10)
[38]     '-- Warten, bis der Server-Socket aktiv ist
[39]         Repeat
[40]           Wait 0.001
[41]         Until ServerSocket.Status = Net.Active
[42]     '-- Konfiguration ausgewählter Steuer-Elemente
[43]         txaLog.Clear()
[44]         cConnetedSockets.Clear()
[45]         btnServiceStartStop.Caption = "Service stoppen"
[46]         MAddOns.SetLEDColor(picBoxOnOff, "green")
[47]         MAddOns.SetNotification("dialog-information", "Hinweis:", ("Der Service wurde gestartet!"))
[48]         LogMessage("\nDer Server läuft auf " & Trim(sIPAddress) & " und lauscht auf Port " & ServerSocket.Port)
[49]         LogMessage("\nDer Service wurde um " & Format(Now(), "hh:nn:ss") & " Uhr gestartet!" & "\n")
[50]      Else
[51]     '-- Verwenden Sie folgende Methode, um alle vom Server gehaltenen Client-Verbindungen zu schließen und
[52]     '-- den Abhörvorgang zu stoppen. Alle Connetion-Sockets werden automatisch in den Status `inaktiv` versetzt.
[53]     '-- Nach dem Anruf der Methode `ServerSocket.Close()` ändert sich der Server-Status auf Net.Inactive.
[54]         ServerSocket.Close()
[55]     '-- Warten, bis der Server-Socket inaktiv ist
[56]         Repeat
[57]           Wait 0.001
[58]         Until ServerSocket.Status = Net.Inactive
[59]     '-- Konfiguration ausgewählter Steuer-Elemente
[60]         btnServiceStartStop.Caption = "Service starten"
[61]         MAddOns.SetLEDColor(picBoxOnOff, "red")
[62]         MAddOns.SetNotification("dialog-information", "Hinweis:", ("Der Service wurde gestoppt!"))
[63]         LogMessage("\nDer Service wurde gestoppt!")
```

```
[64]        txbCurrentClients.Text = "0"
[65]        txaLog.Pos = txaLog.Length
[66]     Endif
[67]
[68] End
[69]
[70] Public Sub ServerSocket_Connection(RemoteHostIP As String)
[71]
[72]     Dim CommunicationSocket As Socket
[73]     Dim sKey As String
[74]
[75] '-- Es wird ein *neuer* Socket auf dem Server erzeugt, wenn der Server eine Client-Verbindungsanfrage akzeptiert hat.
[76] '-- Dieser Socket wird zur Kommunikation zwischen Server und Client verwendet!
[77]     CommunicationSocket = ServerSocket.Accept()
[78] '-- Der neue Socket wird der Liste der aktiven ConnectionSockets auf dem Server hinzugefügt:
[79]     sKey = Hex$(Rand(0, 2 ^ 32 - 1))
[80]     cConnetedSockets.Add(CommunicationSocket, sKey)
[81]
[82]     txbCurrentClients.Text = Str(cConnetedSockets.Count)
[83]     LogMessage(Subst("\nVerbindung auf &1 mit Client &2 &3", RemoteHostIP, sKey, "akzeptiert."))
[84]
[85] End
[86]
[87] Public Sub ServerSocket_Error()
[88]     LogMessage("Es trat ein Fehler auf!")
[89] End
[90]
[91] ''Socket ist der Socket, über den die Kommunikation Server <-> 'Aktiver Client' abgewickelt wird!
[92] Public Sub Socket_Read()
[93]
[94]     Dim sBuffer As String
[95]     Dim LastSocket, Socket As Socket
[96]
[97] '-- Es wird der zuletzt aktive Client ermittelt
[98]     LastSocket = Last
[99] '-- Der Daten-Strom des zuletzt aktiven Clients wird ausgelesen und in `sBuffer` gespeichert
[100]    Read #LastSocket, sBuffer, Lof(LastSocket)
[101] '-- Service:
[102] '-- Die Nachricht des zuletzt aktiven Clients wird mit einem 'ok'-Kommentar an ihn zurück gesendet
[103]    Write #LastSocket, sBuffer & " (ok)", Len(sBuffer & " (ok)")
[104] '-- Die Original-Nachricht des zuletzt aktiven Clients wird auf dem Server protokolliert
[105]    For Each Socket In cConnetedSockets
[106]      If Socket = Last Then
[107]         LogMessage("\nDaten von Client " & cConnetedSockets.Key & " empfangen: " & sBuffer)
[108]      Endif
[109]    Next
[110]
[111] End
[112]
[113] ''Socket ist der Socket, über den die Kommunikation Server <-> 'Aktiver Client' abgewickelt wird!
[114] Public Sub Socket_Closed()
[115]
[116]    Dim sCurKey As String
[117]    Dim Socket As Socket
[118]
[119]    For Each Socket In cConnetedSockets
[120]  '-- Der Client, der sich vom Server abgemeldet hat, wird aus der Liste der aktiven Clients gelöscht
[121]      If Socket = Last Then
[122]         sCurKey = cConnetedSockets.Key
[123]         cConnetedSockets.Remove(sCurKey)
[124]         LogMessage(Subst("\nClient &1 ist offline.", sCurKey))
[125]         Break
[126]      Endif
[127]    Next
[128]
[129]    txbCurrentClients.Text = cConnetedSockets.Count
[130]    txaLog.Pos = txaLog.Length
[131]
[132] End
[133]
[134] '-- PRIVAT ------------------------------------------------------------------------------------------------------
[135]
[136] Private Sub LogMessage(sText As String)
[137]    txaLog.Insert(sText)
[138]    txaLog.Pos = txaLog.Length
[139] End
[140]
[141] Public Sub Form_Close()
[142]
[143]    Dim Socket As Socket
[144]
[145] '-- Wenn das Programm beendet wird, dann müssen wir uns von jedem Client trennen, sonst sorgen wir
[146] '-- für Fehler bei jedem verbundenen Socket und dafür, dass das Programm und auch der Server nur äußerlich
[147] '-- geschlossen werden, der Socket die Verbindung aber immer noch aufrecht erhält.
[148]    For Each Socket In cConnetedSockets
[149]      Socket.Close() '-- Alternative: Close #Socket
[150]    Next
[151] '-- Der Server-Socket wird geschlossen - wenn er existiert und aktiv ist
[152]    If ServerSocket Then
[153]       If ServerSocket.Status = Net.Active Then ServerSocket.Close()
[154]    Endif
[155]
[156] End
[157]
[158] Public Sub btnProgrammEnde_Click()
[159]    FMain.Close()
[160] End
```

24.1.2.1.5 Client-Projekt

Der Modell-Client nutzt den angebotenen Service des Modell-Servers. Der umfangreich kommentierte Quelltext wird wieder vollständig angegeben:

```
[1]  ' Gambas class file
[2]
[3]  Public ClientSocket As Socket
[4]
[5]  Public Sub Form_Open()
[6]
[7]      Dim sIPAddress, sHostname As String
[8]
[9]      Shell ("hostname -I | grep -oE '" & Chr$(40) & "[[:digit:]]{1,3}" & Chr$(92) & "." & Chr$(41) & \
[10]           "{3}[[:digit:]]{1,3}'") To sIPAddress
[11]     Shell ("hostname") To sHostname
[12]     FMain.Caption = "TCPClient | IP-Adresse: " & Trim(sIPAddress) & " | Hostname: " & Trim(sHostname)
[13]     btnSendData.Enabled = False
[14]     txbHost.SetFocus()
[15]     MAddOns.SetLEDColor(picBoxOnOff, "red")
[16]
[17] End
[18]
[19] Public Sub btnConnectDisconnect_Click()
[20]
[21]      If Not txbHost.Text Then
[22]         Message.Warning("<hr>\nEs wurde <b>keine</b> IP-Adresse für den Server angegeben!\n<hr>")
[23]         txbHost.SetFocus()
[24]         Return
[25]      Endif
[26]
[27]      If btnConnectDisconnect.Caption = "Zum Server verbinden" Then
[28]
[29]     '-- Client-Socket erzeugen
[30]         ClientSocket = New Socket As "ClientSocket"
[31]     '-- IP-Adresse des Servers festlegen
[32]         ClientSocket.Host = txbHost.Text
[33]     '-- Port des Dienstes auf dem Server festlegen
[34]         ClientSocket.Port = Val(txbPort.Text)
[35]     '-- Zum Chat-Server verbinden
[36]         ClientSocket.Connect()
[37]
[38]         txaChatHistory.Clear()
[39]
[40]         Do While (ClientSocket.Status <> Net.Connected) And (ClientSocket.Status > Net.Inactive)
[41]           Wait 0.001
[42]         Loop
[43]     '-- Entweder ist der Client-Status = 7 (Net.Connected) oder der Client-Status < 0 -> Fehler!
[44]         If ClientSocket.Status = Net.Connected Then
[45]            btnConnectDisconnect.Caption = "Vom Server trennen"
[46]            MAddOns.SetLEDColor(picBoxOnOff, "green")
[47]            btnSendData.Enabled = True
[48]            txbMessage.Enabled = True
[49]         Else
[50]            Message.Warning("<hr>\nEs ist <b>kein</b> Server erreichbar! IP-Adresse und Port korrekt?\n<hr>")
[51]         Endif
[52]      Else
[53]     '-- Den Client-Socket schließen ...
[54]         ClientSocket.Close()
[55]         Wait 0.1
[56]         If ClientSocket.Status = Net.Inactive Then
[57]            MAddOns.SetLEDColor(picBoxOnOff, "red")
[58]            btnConnectDisconnect.Caption = "Zum Server verbinden"
[59]            btnSendData.Enabled = False
[60]            LogMessage("\nDie Verbindung zum Server wurde durch den Client getrennt!")
[61]         Endif
[62]     Endif
[63]
[64] End
[65]
[66] '' -- Dieses Ereignis wird ausgelöst, nachdem eine TCP/IP-Verbindung vom Client zum Server
[67] '' -- erfolgreich hergestellt wurde. Die Status-Eigenschaft wird auf den Wert `Net.Connected` gesetzt.
[68] Public Sub ClientSocket_Ready()
[69]     LogMessage("\nEine TCP/IP-Verbindung zum Server auf Host " & ClientSocket.Host & " wurde hergestellt!")
[70]     LogMessage("Die Client-Adresse ist " & ClientSocket.LocalHost & ":" & Str(ClientSocket.LocalPort) & Chr(10))
[71]     txbHost.SetFocus()
[72] End
[73]
[74] Public Sub ClientSocket_Read()
[75]
[76]     Dim sBuffer As String
[77]
[78] '-- Der Daten-Strom vom Server wird ausgelesen und in der Variablen `sBuffer` gespeichert
[79]     Read #ClientSocket, sBuffer, Lof(ClientSocket)
[80] '-- Der Inhalt von `sBuffer` wird angezeigt
[81]     LogMessage("◂-- " & sBuffer)
[82]     txbMessage.SetFocus()
[83]
[84] End
[85]
[86] Public Sub ClientSocket_Closed()
[87]     LogMessage("\nDie Verbindung wurde vom Server geschlossen!")
[88]     btnSendData.Enabled = False
[89]     txbMessage.Enabled = False
[90]     btnConnectDisconnect.Caption = "Zum Server verbinden"
```

```
[91]      MAddOns.SetLEDColor(picBoxOnOff, "red")
[92]  End
[93]
[94]  Public Sub btnSendData_Click()
[95]      If Not txbMessage.Text Then Return
[96]      If ClientSocket.Status = Net.Connected Then
[97]         Write #ClientSocket, txbMessage.Text, Len(txbMessage.Text)
[98]         LogMessage("--► " & txbMessage.Text)
[99]      Else
[100]        LogMessage("Es besteht keine Verbindung zum Server!")
[101]     Endif
[102]     txbMessage.Clear()
[103] End
[104]
[105] '-- PRIVAT ------------------------------------------------------------------------------------------------------------------------
[106]
[107] Private Sub LogMessage(sText As String)
[108]     txaChatHistory.Insert(sText & "\n")
[109]     txaChatHistory.Pos = txaChatHistory.Length
[110] End
[111]
[112] Public Sub Form_Close()
[113]     If ClientSocket And If ClientSocket.Status = Net.Connected Then
[114]        ClientSocket.Close()
[115]     Endif
[116] End
[117]
[118] Public Sub btnClose_Click()
[119]     FMain.Close()
[120] End
```

24.1.2.1.6 Beispiel Server-Client-Anwendung

Für den Test der Server-Client-Anwendung werden Server und Client 1 auf dem selben PC gestartet. Es ist aber auch möglich, dass Sie den Server auf PC1 und den Client 1 auf PC2 starten. Wichtig ist nur, dass sich Server und Client 1 für den Test im gleichen Netzwerk befinden. Diese Konstellation wird Ihnen im Kapitel '24.1.2.2 Socket-Projekte' vorgestellt.

Auf dem Server werden die komplette Kommunikation mit jedem Client und die diversen Statusmeldungen protokolliert, während die Clients die Kommunikation sowie einige Statusmeldungen zur TCP-Verbindung anzeigen.

Protokoll:

Abbildung 24.1.2.1.1: Start Server mit 'OK'-Dienst

Der Server startet mit der IP-Adresse 192.168.0.3 und hört auf dem festen Port 8088. Die Port-Eingabe im Textfeld kann mit … freigeschaltet werden, wenn Sie einen anderen Port verwenden wollen. Beachten Sie die Hinweise im Kapitel '24.1.2.1.2 Socket' im Abschnitt, in dem bestimmte Port-Bereiche vorgestellt werden. Tragen Sie beim Client 1 als Adresse die o.a. IP-Adresse 192.168.0.3 oder 'localhost' ein, da sich Server und Client 1 im diesem Test auf dem gleichen Rechner gestartet werden:

Abbildung 24.1.2.1.2: Server-Adresse: localhost

Starten Sie den Client 1 mit 'Zum Server verbinden'. Der Server akzeptiert den Client 1 auf Port *46146*!

Abbildung 24.1.2.1.3: Der Server protokolliert die Verbindungsannahme

Abbildung 24.1.2.1.4: Server-Client-Verbindung kann jetzt genutzt werden

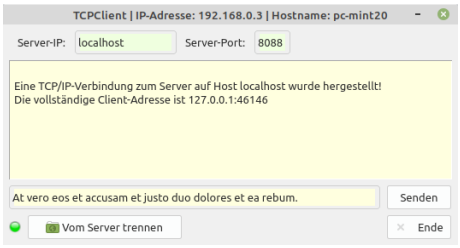

Abbildung 24.1.2.1.5: Texteingabe

Jetzt können Sie Text in das Textfeld eingeben und ihn mit 'Senden' zum Server schicken, dessen Empfang der Server protokolliert:

Abbildung 24.1.2.1.6: Datenempfang (Server)

Anschließend führt der Server seinen Service aus und schickt den empfangenen Text mit der 'OK'-Quittung an den Client 1 zurück.

Abbildung 24.1.2.1.7: Service in Aktion

Danach trennt der Client 1 nach 'Vom Server trennen' die Verbindung zum Server, was der Server mit einer 'Offline'-Meldung und mit einer Änderung der Anzahl der verbundenen Clients quittiert:

Abbildung 24.1.2.1.8: Offline-Meldung vom Client mit dem Port 46146 (Client 1)

Abschließend wird mit 'Service stoppen' der Client geschlossen und somit der angebotene Service gestoppt ...

Abbildung 24.1.2.1.9: Der Service wird gestoppt

… und der Server mit 'Ende' beendet werden.