### 24.1.3.1 UDPSocket-Projekte

In diesem Kapitel werden Ihnen drei Projekte vorgestellt, welche die Klasse *UDPSocket* verwenden.

- Das erste Projekt ist ein Server-Client-Programm. Eingesetzt wird ein UDP-Server, der einen Zitat-Service anbietet, der von UDP-Clients genutzt wird. Sie können kurze und lange Zitate anfordern.
- Das zweite Projekt ruft von einem NTP-Zeit-Server auf dem Port 123 die aktuelle Zeit als Datagramm ab und gibt sie formatiert aus.
- Aus den Quelltexten von Gambas stammt ein Server-Client-Programm, dass vor Allem in Bezug auf die GUI adaptiert wurde. Der UDP-Server stellt den Service bereit, einen an ihn gesendeten Text als reversen Text – aus der Zeichenkette 'Lager' wird etwa 'regaL' – an den Client zurück zu schicken.

Für das zweite Projekt wird der Quelltext im Kapitel vollständig angegeben. Alle drei Projekt-Archive finden Sie im Downloadbereich.

#### 24.1.3.1.1 Projekt 1 – Zitat-Service

Unter https://wiki.pythonde.pysv.org/UDP-Broadcasts finden Sie im Text

"*UDP-Broadcasts können verwendet werden, damit sich Client und Server in einem Subnetz finden, ohne zunächst zu wissen, auf welchen Hosts sie laufen. Der Server öffnet dazu einen Socket, der auf einem bestimmten Port lauscht. Ein Client kann dann einen UDP-Broadcast auf diesen Port an alle Hosts im Subnetz schicken, worauf der Server antworten kann.*"

einen guten Hinweis, wie Sie einen Rundruf im Subnetz realisieren:

```
[1]  Public Sub btnBroadcast_Click()
[2]
[3]      Dim sMessage As String
[4]
[5]      txaShowQuote.Clear()
[6]
[7]  '-- Es wird ein UDP-Socket erzeugt
[8]      UDPClientSocket = New UDPSocket As "UDPClientSocket"
[9]
[10]     If UDPClientSocket.Status <= Net.Inactive Then
[11]        UDPClientSocket.Broadcast = True
[12]        UDPClientSocket.Port = 0 ' Standard-Port für einen UDP-Client
[13]        UDPClientSocket.Bind()
[14]    '-- Standard-Subnetz-Maske im Klasse-C-Netz => 255.255.255.0
[15]        txbTargetHost.Text = sNetzAdresse & ".255" ' IPv4-Broadcast-IP-Adresse
[16]    '-- Ziel-Adressierung: Host und Port
[17]        UDPClientSocket.TargetHost = txbTargetHost.Text
[18]        UDPClientSocket.TargetPort = txbPort.Text
[19]
[20]        Inc Application.Busy
[21]    '-- Ein Datagramm wird an das adressierte Ziel (Server) gesendet
[22]        Write #UDPClientSocket, "broadcast", Len("broadcast")
[23]        Dec Application.Busy
[24]        UDPClientSocket.Broadcast = False
[25]     Endif
[26]
[27]     bBroadcasted = True
[28]
[29]     Wait 0.5
[30]     If Not txaShowQuote.Text Then
[31]       sMessage = "\n\nDer Rundruf wurde von keinem UDP-Socket auf Port "
[32]       sMessage &= txbPort.Text & "\nim lokalen Netzwerk mit der Netzadresse "
[33]       sMessage &= sNetzAdresse & " quittiert!" ""
[34]        LogMessage(sMessage)
[35]        btnClose.SetFocus()
[36]     Endif
[37]
[38] End
```

Kommentar:

- Zeile 10: Nur für einen UDPSocket-Status < 0 können Sie den Port festlegen.
- Mit UDPClientSocket.Broadcast = True in der Zeile 11 und der Ziel-Adressierung in der Zeile 17: UDPClientSocket.TargetHost = txbTargetHost.Text (Broadcast.IP-Adresse) legen Sie fest,

dass alle Datagramme an alle Hosts im Sub-Netz gesendet werden.

- Der vom Client gesendete Text 'broadcast' kann vom gestarteten Server ausgewertet werden – wenn er auf dem zugewiesenen Port gebunden ist. Er sendet dann einen Kommentar zurück, der auch den Hostnamen enthält. Aus der Antwort vom Server kann der Client die IP-Adresse ablesen und den Port, an den der Server gebunden ist.
- In der UDP-Socket-Programmierung hat Port 0 eine besondere Bedeutung – insbesondere bei unixoiden Betriebssystemen – denn Port 0 ist ein reservierter Port in UDP-Netzwerken. Die Befehlsfolge in den Zeilen 12 und 13

```
UDPClientSocket.Port = 0
UDPClientSocket.Bind()
```

wird verwendet, um vom System dynamisch die nächste verfügbare, aktuell nicht verwendete Port-Nummer im Subnetz anzufordern und den Socket an diese Port-Nummer zu binden sowie diesen Port zu öffnen. Damit wird zusammen mit der IP-Adresse eine eindeutige Ziel-Adressierung in den Zeilen 17 und 18 möglich. IP-Dienste, die im gleichen Sub-Netz und daher unter derselben IP-Adresse laufen, werden durch ihre (unterschiedliche) Port-Nummer identifiziert! Starten Sie den UDPServer und den UDPClient auf dem gleichen Computer zum Beispiel unter 127.0.0.1 (localhost) oder wie beim Computer des Autors unter 192.168.2.103 (mint-183), dann wird es Sie nicht verwundern, dass Client und Server mit unterschiedlichen Port-Nummern kommunizieren. Deutlich wird das in folgenden beiden Abbildungen:

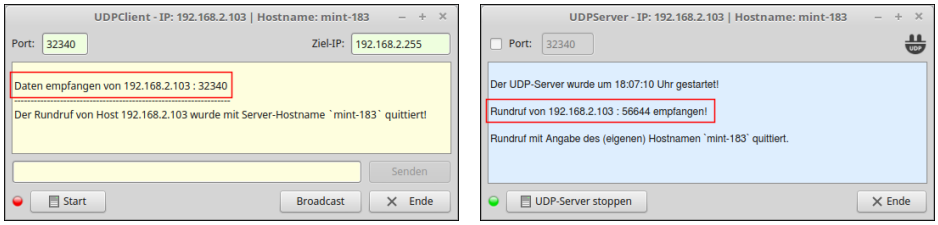


Abbildung 24.1.3.1.1: Kommunikation UDPServer ↔ UDPClient

Nach dem quittierten Rundruf kennt der Client einen aktiven Server auf 192.168.2.103 mit dem Port 32340. Ferner ist erkennbar, dass der Client auf der gleichen IP-Adresse, aber unter dem vom System zugewiesenen Port 56644 mit dem Server kommuniziert.

Um den Zitat-Dienst des Servers zu nutzen und nicht nur eine Mitteilung zu erhalten, dass der angeforderte Service gegenwärtig nicht erreichbar ist, sollten Sie die (deutsche) Sammlung von Zitaten für den Server installieren:

```
sudo apt install fortune-mod      ' Programm
sudo apt-get install fortunes-de  ' Zitat-Sammlung
```

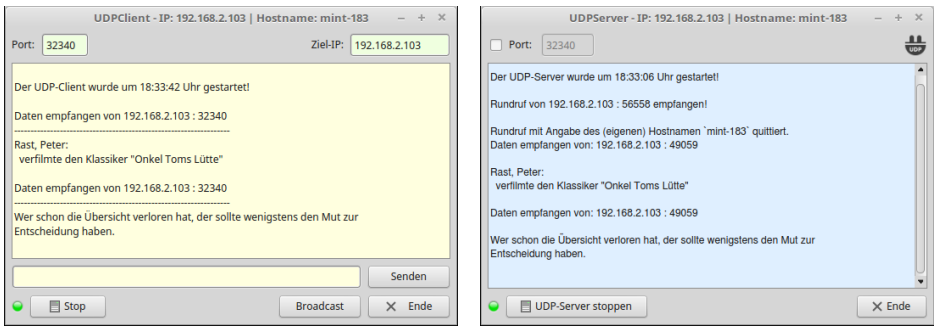


Abbildung 24.1.3.1.2: Server-Client-Programm

Es folgt eine Testbeschreibung für einen Client und zwei Server:

- Zuerst werden ein Client und ein Server auf Computer A gestartet und ein weiterer Server auf einem Computer B (Laptop). Auf dem Laptop ist die Zitat-Sammlung nicht installiert.
- Dann wird vom Client ein Rundruf gestartet, um zu erkunden, ob es Kommunikationspartner auf dem Port 32340 im lokalen Netz gibt. Der Rundruf erfolgt mit Rundruf-IP für Klasse-C-Netze, die nicht in Teilnetze zerlegt sind. Deren Standard-Subnetz-Maske ist 255.255.255.0.

- Es antworten beide Server und geben auch jeweils den Hostnamen mit an. Von beiden Servern kennt der Client somit die IP-Adresse, die für die weitere Kommunikation benötigt wird.
- Danach wird der Client gestartet und die Kommunikation – hier die Nutzung des Zitat-Services von beiden Servern – kann beginnen.
- Es wird zum Beispiel die IP-Adresse des Servers auf dem Computer B angegeben und mit -l (long) ein langes Zitat abgerufen. Der Server antwortet und teilt mit, dass der angeforderte Dienst zur Zeit nicht verfügbar ist. Wenigstens etwas – man wird ordentlich benachrichtigt.
- Nach der Änderung auf die IP-Adresse des UDP-Servers auf Computer A erfolgt der Abruf eines kurzen Zitates mit -s (short). Das Zitat wird vom Server ausgeliefert, von diesem protokolliert und im Client angezeigt.
- Abschließend wird der Client gestoppt. Beide Server bekommen das nicht mit – wie auch, denn bei UDP gibt es keine kontrollierten Verbindungen zwischen den Kommunikationspartnern.

#### 24.1.3.1.2 Projekt 2 – Zeit-Service

Im zweiten Projekt nutzt ein Client den Zeit-Service eines NTP-Servers auf dem reservierten Port 123. Dazu schickt der Client ein leeres Datagramm zum NTP-Server. Dieser quittiert den Empfang des leeren Datagramms mit dem Senden eines Datagramms, dessen Inhalt die aktuelle Zeit enthält:

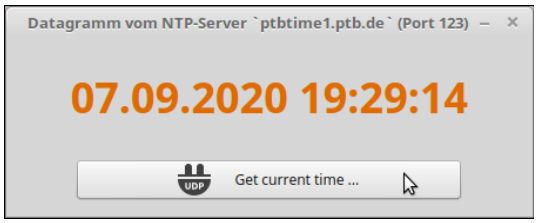


Abbildung 24.1.3.1.3: NTP-Client auf Port 123

Der Quelltext für den Client wird vollständig angegeben:

```
[1]  ' Gambas class file
[2]
[3]  Public UDPClient As UdpSocket
[4]
[5]  Public Sub Form_Open()
[6]      UDPClient = New UdpSocket As "UDPClient"
[7]  '-- Initialisierung des UDP-Sockets
[8]      UDPClient.Port = 0
[9]      UDPClient.Bind()
[10] '-- Vollständige Adressierung:
[11] '-- TimeServerURL: ptbtime1.ptb.de -> IP: 192.53.103.108
[12]     UDPClient.TargetHost = "192.53.103.108"
[13] '-- Für NTP ist der UDP-Port 123 reserviert.
[14]     UDPClient.Targetport = 123
[15]     FMain.Caption = "Datagramm vom NTP-Server `ptbtime1.ptb.de` (Port 123)"
[16] End
[17]
[18] Public Sub UDPClient_Read()
[19]
[20]     Dim sData As String
[21]     Dim i, iVSeconds As Integer
[22]     Dim VSeconds As Long
[23]     Dim CurrentDate As Date
[24]
[25] '-- Auslesen des Zeit-Datagramms vom NTP-Server - gespeichert in `sData`
[26]     Read #UDPClient, sData, Lof(UDPClient)
[27] '-- Auswerten des Zeit-Datagramms -> https://de.wikipedia.org/wiki/Network_Time_Protocol
[28]     For i = 17 To 20
[29]       VSeconds += Asc(Mid$(sData, 37 - i, 1)) * 256 ^ (i - 17)
[30]     Next
[31]     iVSeconds = VSeconds - 2208988800 ' - System.TimeZone
[32]     CurrentDate = DateAdd("01/01/1970", gb.second, iVSeconds)
[33] '-- Formatieren des aktuellen Zeitstempels
[34]     lblDateTime.Text = Format(CurrentDate, "dd.mm.yyyy hh:nn:ss")
[35] End
[36]
[37] Public Sub UDPClient_Error()
[38]
[39]   Select Case UDPClient.Status
[40]     Case Net.CannotBindSocket
[41]       Message.Error(("Unable to Bind to that port"))
[42]     Case Net.CannotCreateSocket
[43]       Message.Error(("System does not allow to create a socket"))
```

```
[44]     Case Net.CannotRead
[45]       Message.Error(("Error Sending Data"))
[46]     Case Net.CannotWrite
[47]       Message.Error(("Error Receiving Data"))
[48]   End Select
[49] End
[50]
[51] Public Sub btnGetTime_Click()
[52] '-- Der NTP-Server reagiert auf den Empfang eines leeren Datagramms mit
[53] '-- dem Senden eines Datagramms, dessen Inhalt den aktuellen Zeitstempel enthält.
[54]
[55] '-- Abschicken eines leeren Datagramms an den NTP-Server -> ' # + 47 mal Nul (=> Chr(0))
[56]     Write #UDPClient, Chr$(35) & String$(47, Chr$(0)), 48
[57] End
[58]
[59] Public Sub Form_Close()
[60]   If UDPClient Then
[61]     If UDPClient.Status > 0 Then UDPClient.Close()
[62]   Endif
[63] End
```

### 24.1.3.1.3 Projekt 2 – Reverse-Service

Das dritte Projekt basiert auf einem Projekt aus dem Quelltext von Gambas, das den Einsatz der Klasse *UDPSocket* demonstriert. Der Service besteht darin, dass eine von einem Client gesendete Zeichenkette vom Server als reverse Zeichenkette an den Client zurück geschickt wird, der den Dienst angefordert hat:

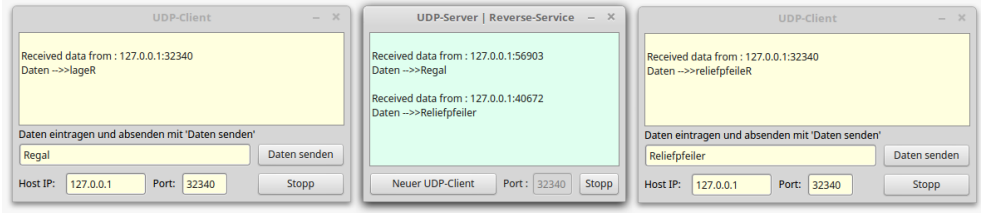


Abbildung 24.1.3.1.4: Hauptprogramm mit der GridView

Beachten Sie die unterschiedlichen Port-Nummern von Client A und von Client B im mittleren Bild, mit denen die Kommunikation mit dem Server realisiert wird!