Ingenieurbüro für Echtzeitprogrammierung

RT-LAN

Netzwerksoftware unter RTOS-UH

Dok-Rev. 3.B vom 28.02.2019

Software-Rev. 15.4 vom 27.02.2017

Inhaltsverzeichnis

Revisionsliste:

Rev.	Datum	Na.	Änderung
1.0	11.07.2000	Ko	Übernahme aus TeX
1.1	22.10.2001	Ha	Verweis auf Pearl-UDP-Beispiel aktualisiert
1.2	14.01.2002	Ha	Default-Timeout FTP-Server nachgetragen
1.3	17.06.2002	Ko	Einrichten einer SLIP Verbindung vom PC
1.4	09.07.2002	Ha	Detaillierungen bei Start NETIO, Passwort-Datei
1.5	01.10.2002	Ko	Fehlermeldung beim UDPREAD in PEARL
1.6	04.02.2003	Ha	TCP_Nreceive ergänzt
1.7	04.03.2003	Ha	Timeout Telnet-Server korrigiert
1.8	05.03.2003	Ko	Übertragungsparameter (Kap. 4.1.1) + BOOTP (Kap. 6) ergänzt.
1.9	11.03.2003	Ko	Beschränkung des Speicherverbrauchs vom Hardwaretreiber
2.0	14.05.2003	Ko	UDP_cleanup beschrieben
2.1	25.06.2003	Kr	Korrektur der Parameterreihenfolge FTP
2.2	08.09.2003	Ko	TCP-Paketgröße korrigiert
2.3	07.01.2004	Ha	NETIO-Start erweitert
2.4	15.01.2004	Ko	Passwortdatei muß genau " , " zwischen User und Passwort enthalten
2.5	01.03.2004	Ko	Fehlermeldungen ergänzt
2.6	25.01.2005	Kr	FTP – Passive Mode ergänzt (Seite 37)
2.7	06.04.2005	Kr	FTP-Server: Passwort Datei: Schreibschutz auf Directorys (Seite 28)
2.8	14.06.2005	Ko	Gateway besser beschrieben
2.9	16.08.2005	Kr	Alive Parameter für NETIO
3.0	03.03.2006	Ko	Beschreibung `tcp_abort` erweitert
3.1	28.06.2006	Ha	Beschreibung Timeout Telnet-Server korrigiert
3.2	05.07.2006	Kr	Passwort Datei mit MD5 verschlüsseltem Passwort
3.3	17.08.2006	Kr	Neue TCP_funktionen ergänzt (ab NETLIB=1.8): TCP_Nsend, TCP_reset, TCP_free_socket, TCP_passive_bind, TCP_wait_listen, TCP_wait_accept
3.4	16.04.2007	Kr	DHCP ergänzt
3.5	15.04.2008	Kr	TCP_wait_listen Parameter Änderung / Beschreibung geändert
3.6	02.03.2010	Ha	Zusätzliche Slip-Parameter, Interpretation des Parameters *ipmask*
3.7	19.07.2010	Ha	Slip: resume-Parameter
3.8	27.01.2011	Ko	Neue Optionen beim Telnet-Server
3.A	15.12.2017	Ko	MEM_LIMIT wird in Bytes gesetzt
3.B	28.02.2019	Kr	TCP_ /passive_bind/wait_listen/wait_accept überarbeitet

1 Urheberrecht und Haftung

Alle Rechte an diesen Unterlagen liegen bei der IEP GmbH, Langenhagen.

Die Vervielfältigung, auch auszugsweise, ist nur mit unserer ausdrücklichen schriftlichen Genehmigung zulässig.

In Verbindung mit dem Kauf von Software erwirbt der Käufer einfaches, nicht übertragbares Nutzungsrecht. Dieses Recht zur Nutzung bezieht sich ausschließlich darauf, daß dieses Produkt auf oder in Zusammenhang mit jeweils **einem** Computer zu benutzen ist. Das Erstellen einer Kopie ist ausschließlich zu Archivierungszwecken unter Aufsicht des Käufers oder seines Beauftragten zulässig. Der Käufer haftet für Schäden, die sich aus der Verletzung seiner Sorgfaltspflicht ergeben, z.B. bei unautorisiertem Kopieren, unberechtigter Weitergabe der Software usw.. Der Käufer gibt mit dem Erwerb der Software seine Zustimmung zu den genannten Bedingungen. Bei unlizensiertem Kopieren muß vorbehaltlich einer endgültigen juristischen Klärung von Diebstahl ausgegangen werden. Dies gilt ebenso für Dokumentation und Software, die durch Modifikation aus Unterlagen und Programmen von IEP hervorgegangen ist, gleichgültig, ob die Änderungen als geringfügig oder erheblich anzusehen sind.

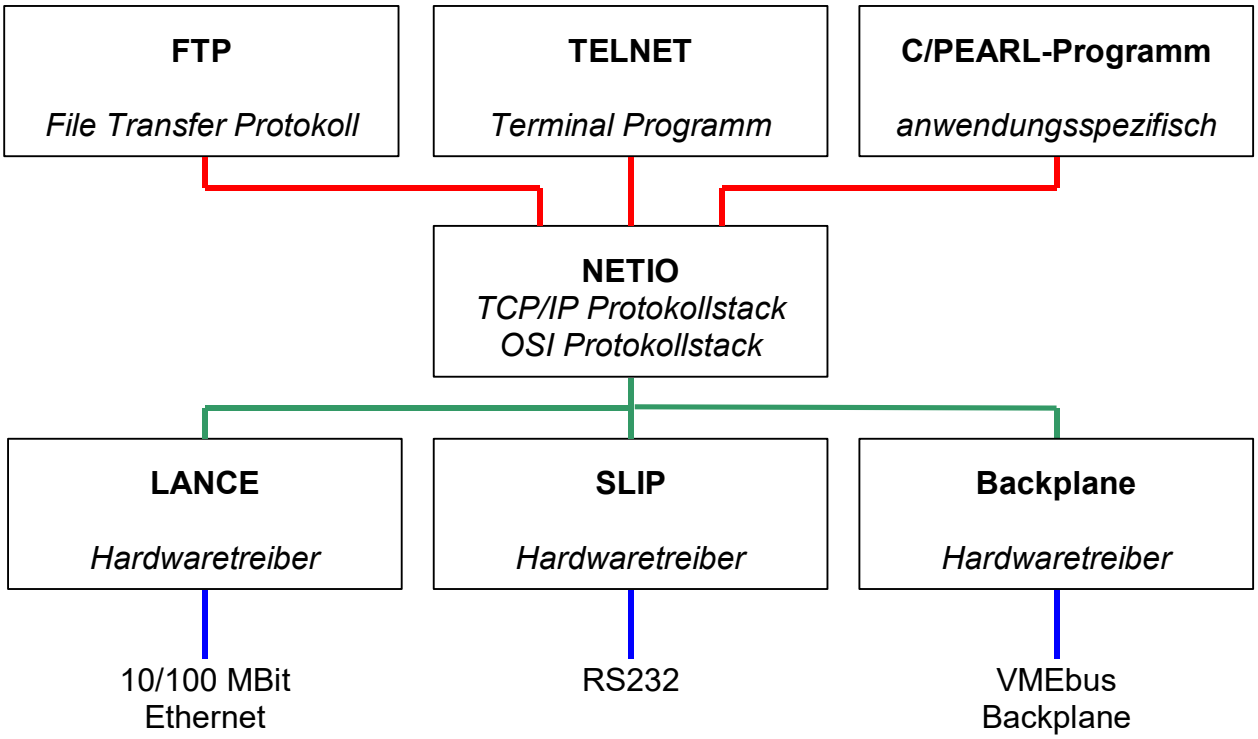
Eine Haftung seitens IEP für Schäden, die auf den Gebrauch von Software, Hardware oder Benutzung dieses Manuskriptes zurückzuführen sind, wird ausdrücklich ausgeschlossen, auch für den Fall fehlerhafter Software oder irrtümlicher Angaben.

Das Einverständnis des Käufers oder Nutzers für den Haftungsausschluß gilt mit dem Kauf und der Nutzung der Software und dieser Unterlagen als erteilt.

2 Netzwerkbetrieb unter RTOS-UH

Diese Dokumentation soll und kann keine allgemeine Beschreibung der verwendeten Netzwerkprotokolle sein. Diese finden Sie in verschiedenen Büchern sowie den RFC's ausführlich erläutert. Vielmehr werden in diesem Dokument die Besonderheiten beim Einsatz unter RTOS-UH beschrieben.

Aufbau der Netzwerkschichten:



Aus dem obigen Bild ist ersichtlich, dass das `NETIO` als zentrale Instanz immer benötigt wird. Im `NETIO` wird das jeweilige Protokoll abgearbeitet. Unter dem `NETIO` wird mindestens ein Hardwaretreiber benötigt, es können aber auch mehrere für unterschiedliche Hardware-Anschlüsse gleichzeitig genutzt werden. Zum Anwender stehen einmal Standard-Programme wie FTP, Telnet oder SAMBA zur Verfügung, zum anderen ist eine Programmierschnittstelle implementiert, die den Zugriff auf die Protokolle TCP, UDP und OSI TP4 erlaubt.

In den folgenden Kapiteln werden alle Teile des RTOS-UH Netzwerkpaketes beschrieben.

3 TCP/IP Protokollstack

3.1 Laden des Protokollstack

Falls der Protokollstack nicht im RTOS-UH EPROM enthalten ist, muß er geladen werden:

```
LOAD NETIO.SR
```

Der Netzwerk-Treiber für die Protokolle UDP/ICMP/TCP wird geladen.

3.1.1 Starten der Netzwerksoftware

Zuerst muß der Protokollstack gestartet werden:

NETIO.NETIO -I=*ip_address* [-N=*xx*][-M=*xx*][-G=*xx*][-V][-m*xxx*][-b*yyy*][-f*r*][-t*zz*][-x][-axx]

Die Parameter haben folgende Bedeutung:

-I=*ip_address*	Eigene IP-Adresse dieser Station z.B: 192.168.10.101 Kann durch später gestartete Hardwaretreiber (z.B. SLIP) übersteuert werden.
-N=*netzwerk*	Abweichender Netzwerkadressraum (darf nicht 0.0.0.0 sein)
-M=*netzmaske*	Abweichende Netzwerkmaske
-G=*gateway*	Gatewayadresse (erfordert -N und -M) (kein Default Gateway! Zum Einrichten eines Defaultgateway ist SET_GW_ADDR zu benutzen, siehe Seite 22).
-V	Ausgabe von Fehlermeldungen, die nicht einem geöffneten Port zugeordnet werden können
-m*xxx*	Von dieser Einstellung ist die MTU (**M**aximum **T**ransfer **U**nit) der TCP-Verbindung abhängig, sie ist defaultmäßig auf 1436 Byte eingestellt, die maximale Länge eines TCP-Telegramms.
-b*yyy*	Mit *yyy* wird die Größe des TCP-Sende- und Empfangspuffers angegeben (Default: 4096 Byte). Bei langsamen Verbindungen wie z.B. SLIP (TCP/IP über serielle Schnittstellen) erhöht eine kleine Buffergröße die Reaktionsgeschwindigkeit. *yyy* wird intern mit 256 multipliziert, d.h. der Defaultwert ist 16.
-f*r*	*r* ist der Retransmit-Faktor. Er legt fest, nach welcher Zeit ein Retransmit durchgeführt wird. Der Default-Wert ist 1, d.h. der erste Retransmit wird nach 512 ms durchgeführt. Bei einer langsamen Verbindung (z.B. Internet) kann es sinnvoll sein, *r* auf einen größeren Wert zu setzten.
-t*zzzz*	Mit dem Parameter *zz* wird das globale Timeout des TCP-Stack gesetzt. Der Default-Wert ist 60 Sekunden. Alle Vorgänge bis auf das Warten auf eine Verbindungseröffnung und das Lesen werden nach dieser Zeit abgebrochen. So wird z.B. ein Sendevorgang incl. des Retransmit nach spätestens *zz* Sekunden mit einer Fehlermeldung beendet. Ebenso wird z.B. ein Socket wieder freigegeben, der geschlossen wurde, ohne das die Gegenseite noch reagiert.

`-x`	Erzeugung von zusätzlichen Debug-Task (`sap`, `sarp`, `show_udp_block`, `show_tcp_block`, `ip_route_table`) zur Verbindungsbeobachtung. Diese Option dient ausschließlich Debugzwecken und sollte nur nach Absprache eingesetzt werden.
`-a`*`zzzz`*	Mit dem Parameter *zz* wird das globale ALIVE-Timeout [in Sekunden] des TCP-Stack gesetzt. Der Default-Wert ist 60 Sekunden. Nach dieser eingestellten Zeit wird bei Inaktivität der Verbindung ein TCP-ALIVE-Paket versendet, auf dieses Paket sollte die Gegenseite reagieren, ansonsten wird nach der doppelten Zeit die Verbindung lokal abgebrochen und der Gegenseite ein TCP-RESET geschickt. Das ALIVE-Timeout dient der Überwachung der Verbindung.

Mit dem Starten des TCP/IP-Protokollstacks ist eine Ansprache des Netzwerkes mit den Protokollen TCP oder UDP (siehe Beispielprogramme) möglich.

Bitte beachten Sie die exakte Schreibweise des Subtask-Namen (`.NETIO`)!

3.2 Die Datei HOSTS

In der `HOSTS`-Datei wird in jeder Zeile die Zuordnung eines Rechnernamens zu einer IP-Adresse hergestellt:

```
; IP-Adresse      Name des Rechners
192.168.10.11     Rechner1
192.168.10.12     server
; dies ist ein Kommentar
192.168.10.100    switch
192.168.100.1     abteilung1
10.0.0.1          abteilung2
```

Die HOSTS-Datei wird z.B. von TELNET und FTP genutzt. Wenn der Rechner, mit dem eine Verbindung hergestellt werden soll, in der HOSTS-Datei eingetragen ist, kann der Name statt der IP-Adresse angegeben werden:

```
TELNET Rechner1
```

Die `HOSTS`-Datei wird auf der RAM-Disk `/R0/` und auf `/H0/ETC/` gesucht.

3.3 Details der Implementierung

NETIO ist Betreuungstask für LDN 17.

Es sollten aus Kompatibilität zu anderen Netzwerkdiensten für eigene Projekte nur Portnummern größer 4096 genutzt werden. Dabei sollte folgende Portvergabe gewählt werden:

PORT = 4096 ... 8191 für Empfangs-Ports (Daten vom Netzwerk)

PORT = 8192 ... 12287 für Sende-Ports (zum Netzwerk)

PORT >= 12288 für lokale Dienste (ohne Netzwerk)

Die Anzahl der gleichzeitig offenen Verbindungen ist nur durch den zur Verfügung stehenden Speicher begrenzt.

3.3.1 Lokaler Modus

Beim lokalen Modus (local delivery) können auf einem Rechner einzelne Tasks über das TCP oder UDP so kommunizieren, als ob sie über das Netz kommunizieren würden. Es ist hierzu notwendig, daß bei der Destination-IP-Adresse die eigene IP-Adresse eingetragen wird. Solche Pakete werden nicht über das Netz transferiert, sondern lokal abgearbeitet.

4 Hardwaretreiber

4.1 10/100 MBit Ethernet

Normalerweise befindet sich der Hardwaretreiber im Betriebssystem integriert und wird automatisch gestartet. Ist der Protokollstack (NETIO) noch nicht gestartet und werden schon Pakete vom Netzwerk empfangen, so wird eine Fehlermeldung ausgegeben, die so oder ähnlich aussieht:

```
>> #ETHDRV WRONG LDN (XIO)
```

Sie können dies als Bestätigung dafür nehmen, dass ein Hardwaretreiber auf Ihrem System vorhanden ist und auch schon Pakete empfangen kann.

4.1.1 Setzen der Übertragungsparameter

Normalerweise müssen die Übertragungsparameter (z.B. 10/100 MBit, Half/Full-Duplex) nicht manuell gesetzt werden. Wenn die Hardware nur eine Übertragungsrate unterstützt (z.B. 10 MBit), so wird diese gesetzt und die Gegenseite – z.B. ein Switch – muss sich entsprechend anpassen. In diesem Fall wird das Setzen der Parameter vom Treiber nicht unterstützt, versuchen Sie es trotzdem, so erhalten Sie eine Fehlermeldung.

Unterstützt die Hardware mehrere Übertragungsarten, so wird versucht, sich automatisch an die Parameter der Gegenseite anzupassen. Normalerweise funktioniert das auch recht gut, es gibt aber Fälle, wo eine Kommunikation zustande kommt, die Übertragungsrate aber mit z.B. 2 KB sehr niedrig liegt. Dann kann eine manuelle Einstellung der Parameter – z.B. die Umschaltung von Fullduplax auf Halfduplex – eine Verbesserung bewirken. Beachten Sie bitte, dass das manuelle Einstellen der Parameter auch dazu führen kann, dass gar keine Kommunikation mehr möglich ist!

Um die Parameter einzustellen, müssen Sie ein CE an den Hardwaretreiber – Standart-LDN=17 – schicken. Dieses CE muß die folgende Struktur im Datenbereich haben:

```
typedef struct set_phy
{
   UWORD cmd   ;  /* must be 9                        */
   UWORD autom ;  /* !=0->automatic, =0 no automatic */
   UWORD speed ;  /* !=0->100 MB, =0->10 MB          */
   UWORD duplx ;  /* !=0->duplex, =0->halfduplex     */
}
   set_phy ;
```

Die `reclen` muß mindestens auf `sizeof( set_phy )` gesetzt werden. Die Struktur ist dann entsprechend den obigen Angaben auszufüllen. Wird Automatik gewählt, so werden die Parameter `speed` und `duplex` ignoriert.

4.1.2 Beschränkung des Speicherverbrauchs

Der Hardwaretreiber generiert für jedes empfangene Ethernetpaket ein CE, dass ans NETIO weitergeleitet wird. Im NETIO werden die Pakete verarbeitet, ggf. weiterverteilt, beantwortet, verworfen usw. Um diese Aufgaben erfüllen zu können, benötigt das NETIO eine bestimmte freie Rechenzeit.

Steht diese Rechenzeit nicht zur Verfügung, da der Rechner anderweitig ausgelastet ist, so kann es passieren, dass der Speicher mit unbearbeiteten CE's vollläuft. Letzlich steht kein freier Speicher mehr zur Verfügung und das System arbeitet nicht mehr. Um diesen Zustand zu verhindern, kann der Speicherverbrauch des Hardwaretreibers beschränkt werden.

Um die max. Speichergröße einzustellen, müssen Sie ein CE an den Hardwaretreiber – Standard-LDN=17 – schicken. Dieses CE muß die folgende Struktur im Datenbereich haben:

```
typedef struct set_mem
{
   UWORD cmd   ; /* must be 11         */
   ULONG limit ; /* mem limit in Bytes */
}
   set_mem ;
```

Die `reclen` muß mindestens auf `sizeof( set_mem )` gesetzt werden. Die Struktur ist dann entsprechend den obigen Angaben auszufüllen. Hat der Treiber das Memorylimit erreicht, so werden weiterhin eingehende Ethernetpakete verworfen, bis wieder genügend Speicher frei ist. Je kleiner Sie die Speichergrenze setzen, desto eher gehen Pakete verloren. Damit sinkt natürlich der Datendurchsatz (TCP) bzw. es gehen dauerhaft Pakete verloren (UDP).

Sie sollten min. 10 KByte pro gleichzeitig offene Verbindung zulassen, wenn Sie nicht größere Performanceverluste in Kauf nehmen möchten. Höhere Werte schaden nicht, sondern erhöhen den Datenduchsatz.

Steht diese Funktion in Ihrem Hardwaretreiber nicht zur Verfügung, erhalten Sie eine Fehlermeldung als Rückgabewert.

4.2 SLIP: Serial-Line-IP

Das **S**erial **L**ine **I**nternet **P**rotokoll ermöglicht Netzwerkverbindungen mit Hilfe des IP-Protokolls über serielle Schnittstellen (RS232 etc.). Hiermit können Netzwerkdienste wie FTP oder Telnet auch über serielle Schnittstellen genutzt werden.

SLIP verwaltet als untere Protokollebene (Hardwaretreiber) nur die Übertragung der Daten über die serielle Verbindung, das eigentliche IP-Protokoll wird vom Protokolltreiber `NETIO` bearbeitet.

4.2.1 Voraussetzungen zum Einsatz von SLIP

Damit SLIP auf einem RTOS-Rechner zum Einsatz kommen kann, müssen folgende Voraussetzungen erfüllt sein:

- Die serielle Schnittstelle, die mit SLIP betrieben werden soll, muß über eine `/CX`*x*`/`-Betriebsart verfügen.

  Die `/CX`*x*`/`-Betriebsart arbeitet wie die übliche `/C`*x*`/`-Betriebsart, allerdings wird `RECLEN = 0` zurückgeliefert, wenn noch kein Zeichen empfangen wurde.

  Das `/CX`*x*`/`-Port arbeitet auf der gleichen LDN wie z.B. das `/A`x`/`-Port, wird jedoch über Drive 14 (hex. `$0E`) angesprochen.

- Die serielle Schnittstelle muß mit 8 Datenbits betrieben werden. Baudrate, Stopbits und Parität müssen wie bei der Gegenseite eingestellt sein. Softwarehandshake kann nicht genutzt werden.

- Die serielle Schnittstelle sollte mit vergrößertem Empfangspuffer (min. 64 Byte) ausgerüstet sein. Dies ist bei allen von IEP gelieferten RTOS-UH Betriebssystemen der Fall.
- NETIO muß als übergelagerter Protokollstack vorhanden sein und **vor** SLIP gestartet werden.

4.2.2 Laden des SLIP-Paketes

Falls das Softwarepaket nicht in den RTOS-EPROMs enthalten ist, muß es nachgeladen werden:

```
LOAD SLIP.SR          der SLIP-Protokolltreiber
```

4.2.3 Starten des SLIP-Treibers

Der SLIP-Treiber wird mit folögendem Aufruf gestartet:

```
slip ipmask [-t=tt] [-c] [(inldn,outldn) [faktor[,resume] [mode
                                [slipldn [D]]]]]
```

Wurde der Protokollstack NETIO nicht vor SLIP gestartet, wartet SLIP eine nur durch den `-t`-Parameter begrenzte Zeit auf den Start des NETIO.

Die Parameter haben folgende Bedeutung:

Parameter	Bedeutung
ipmask 	Mit Hilfe der *ipmask* werden sowohl die eigene IP-Adresse wie auch das über SLIP ansprechbare Subnetz festgelegt. *ipmask* hat den üblichen IP-Adressaufbau (z.B. `192.168.10.x`). Ist die letzte Stelle ≠ 0, so gibt *ipmask* die eigene SLIP-IP-Adresse an. Ist diese Stelle = 0, so wird der Wert aus der korrespondierenden Stelle der beim Start des NETIO angegebenen IP-Adresse verwendet. Dieses Verfahren **sollte nicht** genutzt werden, wenn das NETIO mit `-I=0.0.0.0` gestartet wurde (z.B. um per DHCP eine IP-Adresse zugewiesen zu bekommen), da sich sonst eine unzulässige IP-Adresse ergibt. Die ersten 3 Stellen der *ipmask* legen das über SLIP ansprechbare Subnetz fest.
`-t=`*tt*	Timeout fuer IP-Kommunikation. Findet mehr als *tt* `[min]` keine IP-Kommunikation über SLIP statt, wird SLIP wieder beendet. Default: `-t=0`, d.h. kein Timeout
`-c`	Nur bei Angabe auch von `D`: SLIP wird beendet, wenn über die seriellen Schnittstelle außerhalb eine SLIP -Datenrahmens der String „`NO CARRIER`“ empfangen wurde, z.B. beim Auflegen eines anrufenden Modems. Bei erfolgreichem Verbindungsaufbau (Empfang von mehr als einem IP-Paket) setzt SLIP die globale Umgebungsvariable `SLIP_Axxx_STAT` auf den Wert `CONNECTED`. Diese Variable wird beim Beenden von SLIP wieder gelöscht. Default: `-c` ist nicht aktiv, SLIP ignoriert „`NO CARRIER`“
inldn *outldn*	LDNs des Empfangs- und Sendekanals der von SLIP zu verwendenden seriellen Schnittstelle. Als Ausgabekanal muß ein Duplexkanal (`/Dx/`-Port) angegeben werden. Die Angabe muß ohne umgebende Leerzeichen und direkt hinter der öffnenden runden Klammer erfolgen.

	Die LDNs der Schnittstellen können mit dem RTOS-UH-Befehl „`?-D`“ abgefragt werden. Default: `(2,12)`, d.h. `/A2/` und `/D2/` des RTOS-Rechners `(inldn = 2, outldn = 12)`.
faktor *resume*	*faktor* legt die an der seriellen Schnittstelle zu nutzenden Zeiten für TCP-Timeout, TCP-Retransmit und Einlesepause fest. Das tatsächliche Timeout ergibt sich aus der Multiplikation des jeweiligen Zeitatoms mit dem *faktor*. Die Standard-Zeitatome betragen: TCP-Timeout: 60 sec. Läuft bis zum Ablauf dieser Zeit auf ein gesendetes Paket kein Acknowledge der Gegenseite ein, so wird die Verbindung geschlossen. TCP-Retransmit: 512 msec. Nach dieser Zeit beginnt das TCP mit dem Retransmit der Pakete, für die von der Gegenseite noch kein Acknowledge eingelaufen ist. Einlesepause an der ser. Schnittstelle: 1 msec. SLIP macht zwischen zwei Leseaufträgen von der seriellen Schnittstelle eine Pause der sich hier ergebenden Zeitdauer. *faktor* muß der Baudrate der Übertragungsstrecke (Modemverbindung) entsprechend gewählt werden, höhere Baudraten erlauben einen niedrigeren *faktor*. Wird *faktor* zu klein gewählt, so ergibt sich eine erhöhte Netzlast durch überflüssige Retransmits sowie eine erhöhte Rechnerbelastung durch häufige Einlesevorgänge. Ist *faktor* zu groß gewählt, so wird das Netz nicht ausgelastet, und es können Empfangsfehler durch Überlauf des Empfangspuffers der seriellen Schnittstelle auftreten. Ist die Übertragungsgeschwindigkeit von und zum Modem höher als die Geschwinbdigkeit der Modemstrecken (z.. B. seriell 57600 Buad, Modem-Modem 9600 Baud), so kann über `resume` ein getrennter Faktor zur Bestimmung der Einlesepause vorgegeben werden. `resume` ergibt sich aus der Geschwindigkeit der seriellen Schnittstelle. Default: `faktor = 10`, geeignet für ca. 19200 Baud, `resume = faktor` Beispiel: für eine Verbindung Modem – Modem mit 19200 Baud und eine ser. Schnittstelle mit 57600 Baud sind geeignet: `faktor = 10, resume = 4`

mode	`mode` bestimmt die Betriebsart des SLIP-Protokolls. SLIP kann in folgenden Betriebsarten genutzt werden:
	mode *Betriebsart*
	`0` `8 Bit, SLIP`
	`1` `8 Bit, CSLIP`
	`2` `7 Bit, SLIP6`
	`3` `7 Bit, CSLIP6`
	`8` `adaptiver Betrieb`
	Im adaptiven Betrieb erkennt SLIP bei der ersten Ansprache die Betriebsart der Gegenseite und schaltet die richtige Betriebsart ein.
	Default: `mode = 8`, d.h. adaptiver Betrieb

slipldn	*slipldn* gibt die Basis-LDN zur Kommunikation zwischen SLIP und NETIO an. Über diese LDN kann eine zusätzlichen Anwendung außerhalb der SLIP -Datenrahmen über die serielle Schnittstelle empfangene Zeichen lesen (s.a. Option D). Üblicherweise muß ein Wert angegeben werden, der im aktuellen RTOS-System noch nicht genutzt wird. Sollen mehrere Instanzen von SLIP über unterschiedliche serielle Schnittstellen eines Rechners gestartet werden, so müssen diese auf unterschiedlichen *slipldn* arbeiten. Wird als *slipldn* die Standard-Protokoll-LDN des NETIO angegeben (üblicherweise 17 bzw. hex $11), so wird SLIP als Standardprotokoll des NETIO eingesetzt. Damit entfällt die sich eigentlich aus *ipmask* ergebende Beschränkung eines ansprechbaren Subnetzes, alle nicht von anderen Protokollen bearbeiteten IP-Pakete werden von SLIP über die serielle Schnittstelle bearbeitet. Default: `slipldn = 99`
D	Ermöglicht eine SLIP -Verbindung mit einem PC über ein Nullmodemkabel. SLIP generiert eine zusätzliche Anwendung (`SLIP_DIAL/xx`), die außerhalb von SLIP-Datenrahmen empfangene Zeichen auswertet. Der RTOS-Rechner reagiert dann auf empfangene Anfragen wie folgt: *Empfang* — *Aktion des Slip* `ATD` — sendet connect-string „`CONNECT 19200`“ `AT` — (alle `AT` ohne folgendes „`D`“): sendet „`OK`“ `CLIENT` — sendet „`CLIENTSERVER`“ `NO CARRIER` — nur bei `-c`: Slip wird beendet, sonst: ignoriert SLIP kann damit ohne weitere Hilfsanwendungen über eine PC-Direktverbindung mittels Nullmodem-Kabel oder über ein Modem, das auf automatische Rufannahme programmiert ist, angesprochen werden. **Achtung**!Bei Nutzung dieser Option steht die *slipldn* über für eine weitere Anwendung **nicht** zur Verfügung!. Default: `D` nicht aktiv, `slipldn` kann anderweitig genutzt werden

4.2.3.1 Zusätzliche Steuerungsmöglichkeiten

SLIP legt beim Starten weitere Tasks zur Steuerung des SLIP-Betriebs an. Die Namen dieser Task enden mit `_A`*xxx*, wobei *xxx* die dreistellige, dezimale Angabe der *inldn* ist. Führende Nullen werden mit ausgegeben.

Es stehen folgende Tasks zur Verfügung:

Taskname	Bedeutung
`SLIP_STOP_A`*xxx*	beendet den SLIP-Betrieb dieser Instanz. Nach einer Abschlußmeldung beendet der SLIP-Treiber seine Arbeit und stellt die serielle Schnittstelle wieder zur üblichen Nutzung zur Verfügung.
`SLIP_MODE_A`*xxx*	dient zur gezielten Einstellung der Betriebsart bei laufendem Treiber. Dies ist insbesondere sinnvoll, wenn die Gegenstelle wechselt und der Treiber nicht gestoppt und neu gestartet werden soll.
`SLIP_SHOW_A`*xxx*	gibt einige Informationen über den Zustand des Treibers aus.

4.2.4 Beispiel für den SLIP-Aufruf

Mit den Aufrufen

```
NETIO.NETIO -I=192.168.10.142
SLIP 192.168.30.0 (0,11) 4 0 99 D
```

wird eine SLIP-Instanz mit den folgenden Eigenschaften gestartet:

- Die benutzte serielle Schnittstelle ist die `/A1/`.
- Die eigene IP-Adresse (über SLIP) ist `192.168.30.142`.
- Die SLIP-Gegenstelle kann über eine IP-Adresse im Bereich 192.168.30.1...192.168.30.254 angesprochen werden. Selbstverständlich muß die Gegenstation eine andere IP-Adresse als die eigene Station besitzen.
- Der *faktor* 4 wurde für den Betrieb der Schnittstelle mit 38400 Baud gewählt.
- SLIP wird mit einem 8 Bit SLIP-Protokoll gestartet.
- Die *slipldn* ist 99.
- Es werden die Tasks `SLIP_STOP_A000`, `SLIP_MODE_A000` und `SLIP_SHOW_A000` generiert.
- Es wird auch die Task `SLIP_DIALER/`*xx* erzeugt, damit die Verbindung über ein Nullmodem-Kabel vom DFÜ-Netzwerk des PC gestartet werden kann.

4.2.5 Betrieb einer SLIP-Verbindung über Modem

Soll über eine serielle Schnittstelle mit Modem SLIP gefahren werden, so ist es ggf. erforderlich, die Schnittstelle gleichzeitig zur Kontrolle des Modembetriebs zu nutzen.

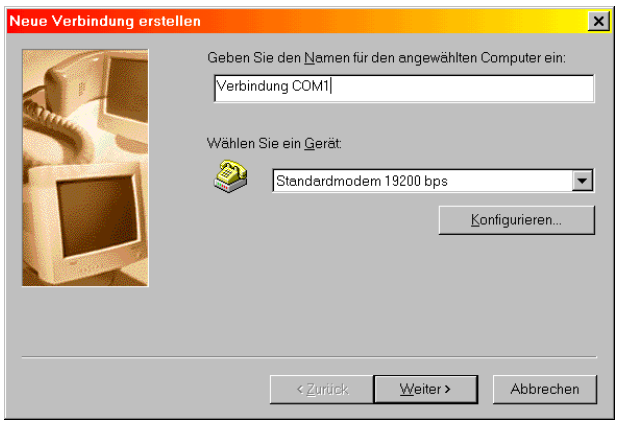
Zu diesem Zweck stellt SLIP unter *slipldn* eine Datenstation bereit, von der alle Empfangsdaten, die nicht zum SLIP-Protokollrahmen gehören, sowie mit einem Timeout abgebrochene SLIP-Telegramme, gelesen werden können. Als Timeout für SLIP-Telegramme wird der Wert von *faktor* in Sekunden verwendet.

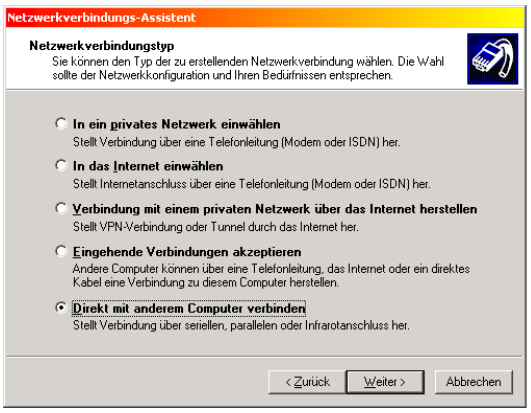
Von dieser Datenstation kann nur gelesen werden. Sie verhält sich wie eine gepufferte, serielle Schnittstelle (`/B`*x*`/`-Betrieb). Die Puffergröße entspricht der maximalen SLIP-Paketlänge.

4.2.6 Einrichten einer Verbindung PC <-> RTOS-UH

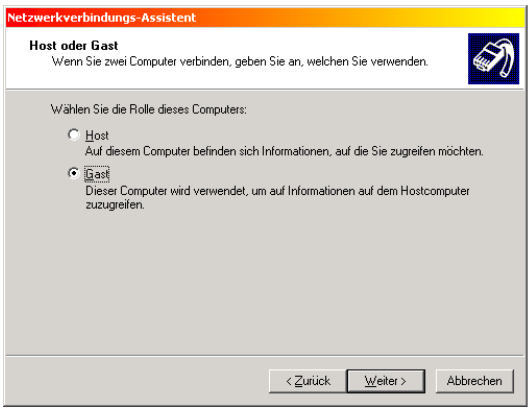
Sie können über ein Nullmodem-Kabel eine Verbindung von einem PC zu einem RTOS-UH Rechner herstellen. Damit die Verbindung zu stande kommt, müssen zuerst die Parameter (Baudrate usw.) auf beiden Seiten gleich eingestellt werden. Testen Sie die Verbindung vor dem Starten des SLIP mit einem Terminalprogramm (z.B. RTW). Wenn Sie auf diese Weise mit dem RTOS-UH-Rechner kommunizieren können, starten Sie SLIP wie oben angegeben mit der Option `D`.

Auf der PC-Seite müssen Sie jetzt eine entsprechende DFÜ-Verbindung herstellen. Links sehen Sie die Screenshots für W98, rechts für W2000.

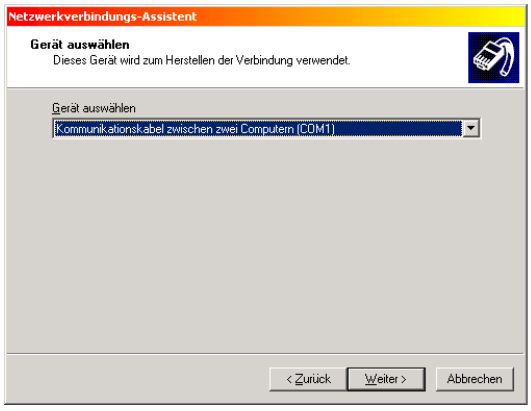




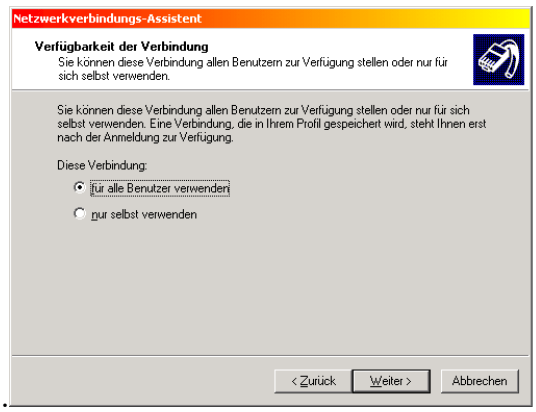
Wählen Sie unter W98 ein beliebiges Standardmodem, unter W2000 können Sie die Direktverbindung wählen.



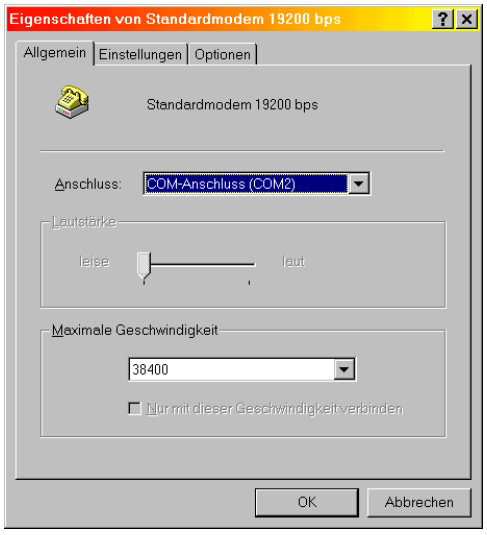
Die Werte für die Rufnummern sind unwichtig. Unter W2000 müssen Sie Gast wählen, um auf den RTOS-Rechner zugreifen zu können.

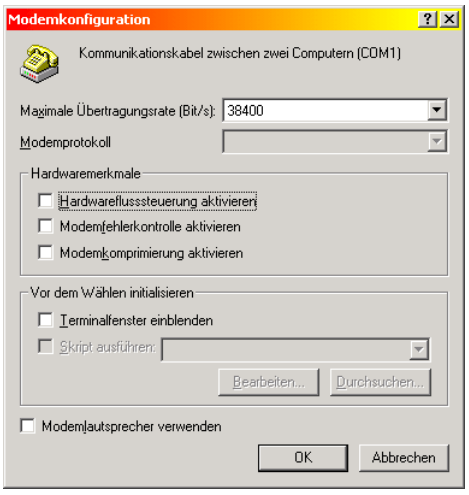


Wählen Sie die Kommunikation über serielle Verbindung.

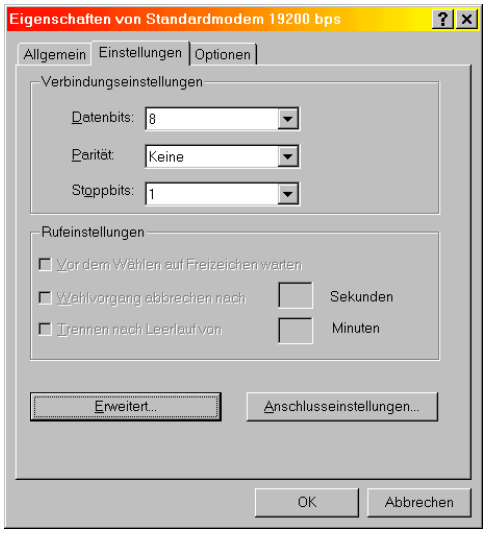


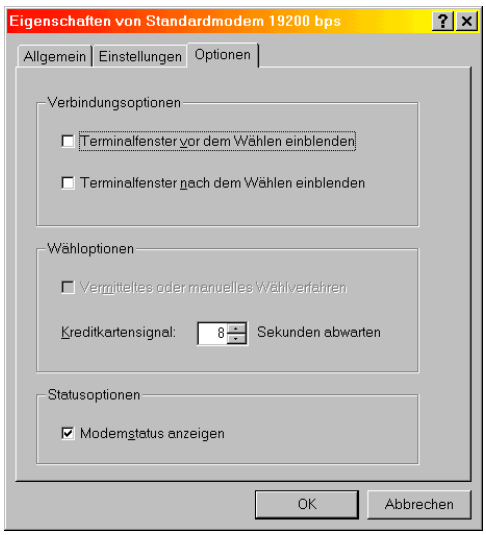
Die Benutzerverwaltung unterliegt Ihren individuellen Anforderungen. Nun müssen Sie nur noch einen Namen wählen und die Verbindung fertig stellen. Die Konfiguration erfolgt über die Eigenschaften (rechte Taste) der Verbindung:

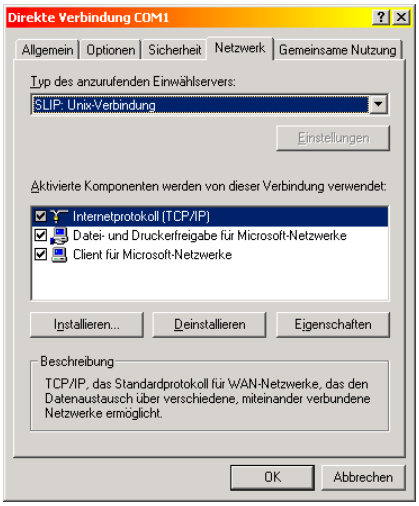




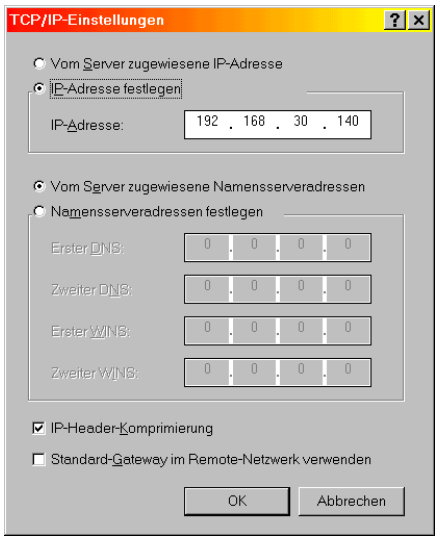
Wählen Sie die COM-Schnittstelle und die Baudrate der Verbindung. Die Baudrate muß der Baudrate des RTOS-UH-Rechners entsprechen.

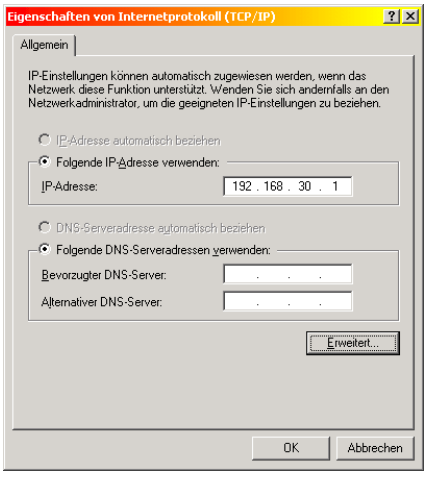




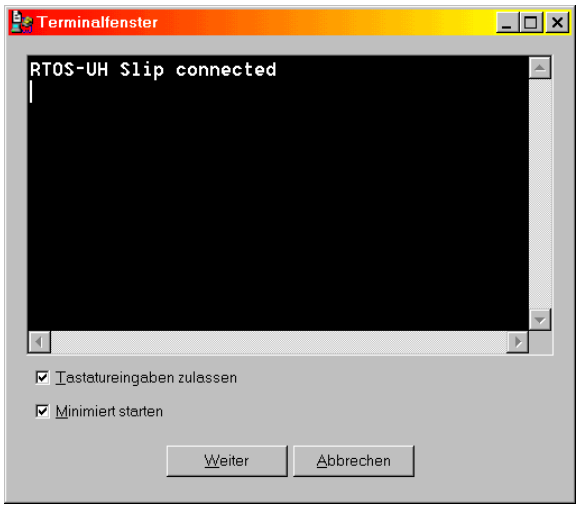


Nun müssen Sie für die Verbindung den Typ SLIP wählen und danach die TCP/IP Einstellungen anpassen. Wählen Sie ein feste IP-Adresse aus dem zulässigen Bereich:





Die weiteren Einträge können mit den Standardwerten besetzt bleiben. Wenn Sie nun die Verbindung starten und das Terminalfenster eingeblendet haben, erhalten Sie folgende Meldung:



Nach dem Klick auf „Weiter“ steht die Verbindung zur Verfügung. In der Statusleiste sehen Sie die etablierte Modemverbindung. Mit einem Ping können Sie überprüfen, ob die Verbindung funktioniert.

Können Sie keine Verbindung aufnehmen, so testen Sie zuerst die serielle Verbindung ohne SLIP (Terminal) und prüfen Sie auch, ob das SLIP mit der Option `D` gestartet wurde.

5 Einrichtung eines Gateways unter RTOS-UH

5.1 Aufgabe eines Gateway-Rechners

In einem Netzwerk ist ein Rechner durch die Angabe von IP-Adresse und Netzmaske eindeutig bestimmt. Er kann nur mit Rechnern kommunizieren, die im gleichen Subnetz liegen. Soll eine Kommunikation mit Rechnern aus anderen Subnetzen stattfinden, ist eine Vermittlungsstation nötig, ein Gateway. An dieses werden alle TCP/IP-Pakete geschickt, die nicht im eigenen Subnetz liegen. Der Gateway-Rechner leitet sie dann – ggf. über weitere Gateways – an den Zielrechner weiter.

Ein kleines Beispiel soll das gesagte verdeutlichen:

Rechner1 soll die IP-Adresse `192.168.10.11` haben. Für diese IP-Adresse muß die Netzmaske auf `255.255.255.0` gesetzt werden, da es sich um ein Klasse-C Netz handelt. Damit können alle Rechner mit einer IP-Adresse von `192.168.10.`*xxx* erreicht werden, d.h. im Subnetz `192.168.10.0`. Soll nun ein *Rechner2* mit der Adresse `192.168.100.12` angesprochen werden, muß ein Gateway die Vermittlung zwischen den beiden Sub-Netzen übernehmen. Dazu muß *Rechner1* die Adresse des Gateway-Rechners mitgeteilt werden.

5.2 Setzen der Gateway-Adresse

Mit Hilfe des Bedienbefehles kann die Gateway-Adresse gesetzt werden.

`SET_GW_ADDR -N=`*net_addr* `[-M=`*net_mask*`] -G=`*gateway_addr* `[-I=`*ip_addr*`] [-D=`*ldn*`] [-E]`

`-N=`*net_addr*	Subnetzadresse	z.B. `192.168.10.0`
`-M=`*net_mask*	Netzmaske	z.B. `255.255.255.0`
`-G=`*gateway_addr*	Gatewayadresse	z.B. `192.168.10.99`
`-I=`*ip_addr*	Eigene IP-Adresse	z.B. `192.168.10.11`
`-D=`*ldn*	LDN des physikalischen Interfaces	z.B. `17`
`-E`	Löschen des Eintrages in der Routingtabelle	

Mit der Subnetzadresse wird das Subnetz klassifiziert, für den dieser Routingeintrag gelten soll. Die Netzmaske sollte entsprechend der Subnetzklasse (A, B oder C) gesetzt werden.

Wird eine Gateway-Adresse ungleich einer eigenen IP-Adresse eingetragen, so nehmen die IP-Pakete grundsätzlich der Weg über den Gateway-Rechner. Der Gateway-Rechner muß also in diesem Netzt verfügbar sein. Entspricht die Gateway-Adresse einer eigenen IP-Adresse, werden die Pakete direkt über das entsprechende physikalische Interface des lokalen Rechner versandt.

Mit dem Parameter `-D=` wird die LDN des physikalischen Interfaces auf dem RTOS-Rechner angegeben.

Wird der Protokollstack mit dem Aufruf:

```
NETIO.NETIO -I=192.168.10.11
```

gestartet, entspricht dies einem Aufruf von SET_GW_ADDR mit folgenden Parametern:

```
SET_GW_ADDR -N=192.168.10.0 -M=255.255.255.0 -G=192.168.10.11
            -I=192.168.10.11 -D=17
```

Auf diesem Rechner ist also kein Gateway eingetragen. Alle IP-Pakete mit IP-Adressen im Adreßraum von 192.168.10.0 bis 192.168.10.255 werden auf dem physikalischen Interface mit der LDN=17 ausgegeben. Alle anderen IP-Adressen können nicht ausgegeben werden, da keine Regel für den Weg existiert.

Falls dieser Rechner einen Rechner aus dem Netz 192.168.100.*xxx* erreichen soll und als Vermittlungsstation ein Gateway-Rechner mit der IP-Adresse 192.168.10.99 zur Verfügung steht, muss folgender zusätzlicher Eintrag gemacht werden:

```
SET_GW_ADDR -N=192.168.100.0 -G=192.168.10.99
```

Um einen Routingeintrag zu löschen muß hinter der Subnetzadresse der Parameter **-E** eingegeben werden.

```
SET_GW_ADDR -N=192.168.100.0 -E
```

Es wird der Eintrag für das Netz 192.168.100.0 gelöscht.

5.3 Anzeige der Routingtabelle

Der Bedienbefehl SHOW_GW_ADDR gibt die aktuelle Routingtabelle aus:

```
--- Active Routes 00F7CCB0
 Network Address   Network Mask  Gateway Address      Interface  LDN  RF
    192.168.10.0  255.255.255.0    192.168.10.11  192.168.10.11   17   1
   192.168.100.0  255.255.255.0    192.168.10.99  192.168.10.11   17   1
    192.168.55.0  255.255.255.0    192.168.55.11  192.168.55.11   99   1
--- End of Route Table
```

Dieser Rechner hat zwei physikalische Interfaces, deren Betreuungstasks auf LDN 17 und LDN 99 liegen. Die IP-Adresse für das Interface auf LDN 17 ist 192.168.10.11, für das Interface auf LDN 99 ist es 192.168.55.55. Um Rechner im Subnetz 192.168.100.0 erreichen zu können, werden die IP-Pakete über den Gateway-Rechner mit der IP-Adresse 192.168.10.99 geleitet.

Damit sind alle Rechner in den Subnetzen 192.168.10.0, 192.168.100.0 und 192.168.55.0 erreichbar.

Um ein IP-Paket zu routen wird die Routingtabelle solange von oben nach unten durchsucht, bis ein Eintrag mit einer passenden Route gefunden wird.

5.4 Default Gateway

An ein Default-Gateway werden alle IP-Pakete geleitet, die nicht im eigenen Subnetz liegen. Dies ist z.B. nötig, wenn auf das Internet zugegriffen werden soll. Um einen Eintrag für ein Default-Gateway zu erzeugen, ist ein Eintrag mit -N=0.0.0.0 zu erzeugen.

6 BOOTP

BOOTP ermöglicht es, die IP-Adresse, Maske, Gatewayadresse usw. von einem BOOTP-Server zu bekommen. Somit muss der Rechner nicht erst konfiguriert werden, um in ein bestehendes Netzwerk integriert zu werden.

Um das BOOTP-Protokoll sinnvoll einsetzen zu können, muss das NETIO mit der IP-Adresse `0.0.0.0` gestartet werden. BOOTP ist mit dem folgenden Aufruf zu starten:

`BOOTP [-I=`*`own_ip`*`] [-S=`*`server_ip`*`] [-F=`*`bootfile`*`] [-T=`*`timeout`*`] [-P=`*`path`*`]`

Die Parameter haben folgende Bedeutung:

own_ip	eigene IP-Adresse (default = `0.0.0.0`)
server_ip	IP-Adresse des BOOTP-Servers. Es wird auf die Antwort dieses Servers gewartet, sonst wird die Antwort des Servers, der sich als erstes meldet, genommen.
bootfile	Hier kann dem Server ein gewünschtes Bootfile mitgeteilt werden, *bootfile* wird ggf. vom Server überschrieben.
timeout	Die Wartezeit auf den Server kann eingestellt werden, die Defaultzeit beträgt 60s.
path	Pfad des bootfiles, auch dieser Wert wird ggf. vom Server überschrieben.

Wenn ein Server gefunden wurde, liefert der Aufruf von BOOTP den Status OK zurück, wurde kein Server gefunden, wird der Status "nicht OK" zurückgeliefert.

Wurde ein Server gefunden, werden die Rückgabewerte, die der Server geliefert hat, automatisch gesetzt. Folgende Werte werden beeinflußt: Netzwerk, Netzwerkmaske, Gatewayadresse und eigene IP-Adresse. Die Netzwerkmaske wird so gesetzt, dass ein Gatewayrechner auch erreichbar ist.

BOOTP gibt die empfangenen Daten über STD-Out aus, d.h. beim automatischen Start über die Schnittstelle `A1`. Sind Sie an den Daten nicht interessiert, starten Sie BOOTP mit

```
o /nil/bla bootp...
```

Möchten Sie die Daten noch weiter verarbeiten, so kann die Umlenkung in eine Datei sinnvoll sein:

```
o /r0/bootp.prt bootp...
```

Die folgenden Zeilen werden ausgegeben:

```
CLIENTIP=own_ip
SERVERIP=server_ip
ROUTERIP=gateway_ip
BOOTPATH=path
BOOTFILE=bootfile
SERVNAME=server_name
HOSTNAME=own_name
```

Ab `ROUTERIP` werden die Werte nur ausgegeben, wenn Sie vom BOOTP-Server geliefert wurden. Antwortet kein BOOTP-Server erhält man die folgende Ausgabe:

```
<ERROR> No Boot-Reply seen in xxx msec
```

7 DHCP

DHCP ermöglicht es, die IP-Adresse, Maske, Gatewayadresse usw. von einem DHCP-Server zu bekommen. Somit muss der Rechner nicht erst konfiguriert werden, um in ein bestehendes Netzwerk integriert zu werden.

Um das DHCP-Protokoll sinnvoll einsetzen zu können, muss das NETIO mit der IP-Adresse `0.0.0.0` gestartet werden. DHCP ist mit dem folgenden Aufruf zu starten:

```
DHCP
```

Wenn ein Server gefunden wurde, liefert der Aufruf von DHCP den Status OK zurück, wurde kein Server gefunden, wird der Status "nicht OK" zurückgeliefert.

Wurde ein Server gefunden, werden die Rückgabewerte, die der Server geliefert hat, automatisch gesetzt. Folgende Werte werden beeinflußt: Netzwerk, Netzwerkmaske, Gatewayadresse und eigene IP-Adresse. Die Netzwerkmaske wird so gesetzt, dass der Gatewayrechner auch erreichbar ist.

Zusätzlich setzt DHCP folgende globale Environment Variablen:

OWN_IPADDR	mit den IP-String der vom DHCP-Server zugewiesenen IP-Adresse
OWN_NETWORK	mit den IP-String des vom DHCP-Server zugewiesenem Network
OWN_SUBNET	mit den IP-String des vom DHCP-Server zugewiesenem Subnet
DHCP_SERVER	mit dem IP-String des DHCP-Servers
DNS_SERVER	mit dem IP-String des DNS-Servers (wenn geliefert)
GATEWAY	mit dem IP-String des Gateways (wenn geliefert)

Als Abschlussmeldung wird folgender String auf die aufrufende Konsole ausgegeben:

```
DHCP: <INFO> Dyn-IP: xxx.xxx.xxx.xxx
```

Antwortet kein DHCP-Server erhält man die folgende Ausgabe:

```
DHCP: <ERROR> No DHCP-Reply seen
```

8 Terminalverbindungen mit TELNET

8.1 Laden des Paketes

Ist das Softwarepaket nicht in den RTOS-EPROMs enthalten, muß es erst vom Massenspeicher nachgeladen werden. Es stehen ein Telnet-Server (ermöglicht den Zugriff auf ein RTOS-System von einem externen Rechner) und ein Telnet-Client (ermöglicht den Zugriff auf einen externen Rechner von einem RTOS aus) zur Verfügung.

```
LOAD TSERVER.SR      der TELNET-Server
LOAD TELNET.SR       der TELNET-Client
```

8.2 TELNET-Server

Der TELNET-Server wird mit folgendem Befehl gestartet.

```
TEL_SERVER [-p=pass_wort_datei] [-t=time_out]
```

Der RTOS-Rechner ist nun unter der beim Start vom NETIO angegebenen *ip_address* mit einem TELNET-Client erreichbar. Der TEL_SERVER stellt dem Anwender einen ganz normalen RTOS-User zur Verfügung. Die Kommunikation wird über eine TCP-Verbindung geführt. Es wird die Port-Nummer 23 (Standard Telnet-Port) belegt.

RTOS-UH stellt max. 5 Telnet-Verbindungen gleichzeitig über das Netz zur Verfügung. Alle weiteren Verbindungswünsche werden mit einer Fehlermeldungen abgewiesen.

Mit der Option `-t` kann die Timeout-Zeit eingestellt werden. Wenn auf einer Telnet-Verbindung keine Aktivität (Datenaustausch, TCP-ALIVE-Pakete bleiben unberücksichtigt!) feststellbar ist, schließt der Server die Verbindung nach der angegebenen Timeout-Zeit. Sie kann im Minutenraster eingestellt werden. Wird die Timeout-Zeit auf 0 gesetzt, findet keine Zeitüberwachung statt (default). Dabei ist zu beachten, dass ein TCP-Port belegt – und damit nicht mehr nutzbar – liegen bleiben kann, wenn der Telnet-Client vom Netz getrennt wird, ohne dass er sich abmelden konnte.

8.2.1 TELNET-Server Passwort-Datei

Mit der Option `-p` kann dem Telnet-Server eine Passwort-Datei zugeteilt werden. In dieser sind Usernamen und Passwörter für den jeweiligen User verzeichnet. Es sollte immer der komplette Pfad der Passwortdatei angegeben werden.

Beispiel:

```
TEL_SERVER -p=/H0/MY_PASSWD
```

8.2.1.1 TELNET-Server Klartext Passwort

Die Datei MY_PASSWD (mit Klartext Passwort) habe folgenden Inhalt:

```
GUEST , GUEST    * User GUEST mit Passwort GUEST (Klartext)
root , Master    * User root mit Passwort Master (Klartext)
Karl , Xy4bla    * User Karl, Passwort Xy4bla (Klartext)
```

Es dürfen die User GUEST, root und Karl über Telnet auf den Rechner zugreifen. Es kann die gleiche Passwortdatei wie für den FTP-Server verwendet werden. Bitte beachten Sie, dass zwischen Nutzernamen und Passwort ein Leerzeichen, ein Komma und ein Leerzeichen stehen müssen und die Datei keine Leerzeilen enthalten darf!

8.2.1.2 TELNET-Server MD5 Passwort

Es besteht ab TELSRV=3.0 die Möglichkeit in der Passwortdatei das Passwort MD5 zu verschlüsseln, zur Verschlüsselung ist ein MD5TOOL notwendig, daß den Schlüssel errechnet. Bei Bedarf können Sie ein entsprechendes Programm bei uns erhalten.

Die Datei MY_PASSWD (mit MD5 verschlüsseltem Passwort) habe folgenden Inhalt:

```
GUEST : Ex6W83924c79d793461accf92cb9b883323a     * User GUEST mit  MD5 für GUEST
root : Af6S554892f7c15603ee8a3469f99345cf60      * User root mit  MD5 für Master
Karl : Yx0S8cfb2f2e965c203637aecea60aedd2ca      * User Karl mit  MD5 für Xy4bla
```

Es dürfen die User GUEST, root und Karl über Telnet auf den Rechner zugreifen. Es kann die gleiche Passwortdatei wie für den FTP-Server verwendet werden. Bitte beachten Sie, dass zwischen Nutzernamen und Passwort ein Leerzeichen, ein Doppelpunkt und ein Leerzeichen stehen müssen und die Datei keine Leerzeilen enthalten darf!

Sie dürfen in einer Passwortdatei sowohl Passworteinträge im Klartext als auch MD5-Passworteinträge verwenden.

8.2.1.3 Bedienaufruf beim Start der Telnet Session

Ab V4.0 kann beim Start der Telnet Session sofort ein Bedienbehl abgesetz werden, die Kommandoebene des Systems ist dann nicht mehr erreichbar. Damit kann z.B. eine Fernbedienung des Rechners erfolgen, ohne dass der angemeldete Nutzer Zugriff auf weitere Bedienkommandos bekommen kann.

Die Passwort-Datei wird dazu um die folgenden Einträge erweitert:

Hinter *User , Passwort* steht ein " ;", dann folgt der Befehl, der am Ende mit einem "*" begrenzt wird. Beispiel:

```
GUEST , GUEST ; S*
Root , Master
Karl , Xy4bla ;FERNBED*
```

Beim User GUEST wird nur der Bedienbefehl S abgesetzt, danach ist die Telnet Session beendet. Beim Einloggen als Root steht die Bedienshell uneingeschränkt zur Verfügung, alle Aktionen sind erlaubt. Beim Einloggen als Karl wird die Task FERNBED gestartet, die dann z.B. ein Bedienmenue zur Verfügung stellt.

Damit noch ein Parameter übergeben werden kann, besteht die Möglichkeit, das Passwort durch ein * zu ersetzen, dann wird das eingegebene Passwort als Parameter an den Befehl gehängt:

```
Karl , * ;FERNBED -G=*
```

Bei Eingabe eines Passwortes z.B. als HuHu lautet der Aufruf dann FERNBED –G=HuHu.

8.3 TELNET-Client

Um eine Telnet-Session zu einem anderen Rechner zu starten, muss der Telnet-Client mit folgendem Kommando aufgerufen werden:

```
TELNET ip_address|hostname [-t terminal_type]
```

hostname kann nur benutzt werden, wenn auf dem RTOS-Rechner eine HOSTS-Datei vorhanden ist. Sie muß auf /R0/ oder /H0/ETC/ liegen.

terminal_type legt die verwendete Terminalemulation fest. Sie sollte mit dem verwendeten Terminal übereinstimmen, sonst kann sowohl die Ausgabe unleserlich werden als auch die Eingabe (Cursortasten) unbrauchbar sein. Wenn der Parameter fehlt, wird der Standard-Mode TVI925 gewählt. Eine andere mögliche Eingabe wäre z.B. VT100.

Mit Absetzen des Kommandos wird versucht auf dem TCP-Port 23 des fremden Rechners einen Telnet-Server anzusprechen. Kann eine Verbindung hergestellt werden, wird die Meldung

```
TELNET Version x.y ready
```

ausgegeben. Ist der angewählte Rechner nicht erreichbar, so wird die Meldung

```
>>TELNET Could not reach Remote: ip_adr (stopped)
```

ausgegeben. Beim Beenden der Telnet-Verbindung erfolgt die Ausgabe von:

```
>>TELNET stopped
```

Ist ein Telnet-Server erreichbar, erfolgt eine Betriebssystem-abhängige Meldung.

8.3.1 Verbindung zu einem RTOS-Telnet-Server

Wird die Telnet-Verbindung zu einem RTOS-Rechner hergestellt, ist je nach Passwortdatei eine Anmeldung erforderlich oder nicht. Der User muß – wie unter RTOS-UH üblich – mit Ctrl A "geweckt" werden. Der RTOS-Prompt wird durch die Ausgabe von *ip_adresse*> ersetzt.

8.3.2 Verbindung zu einem nicht RTOS-Telnet-Server

Ist die Verbindung zu einem Telnet-Server hergestellt, der auf einem anderen Betriebssystem läuft, wird üblicherweise die dortige Überschriftsmeldung ausgegeben. Danach wird meistens zur Eingabe von Username und Passwort aufgefordert. Ist auch diese Hürde erfolgreich genommen, sollte der Rechner so bedient werden können wie über ein normales Textterminal.

8.3.3 Suspendieren und Beenden der Telnet-Session

Mit ESC ESC S kann das Telnet suspendiert werden. Es erfolgt die Ausgabe von

```
TELNET/xxxx: ( suspended )
```

Das Telnet wird suspendiert und Ihre Eingaben werden wieder vom RTOS-Bedieninterpreter entgegen genommen. Mit einem

```
c TELNET/xxxx
```

kann das Telnet fortgesetzt werden.

Mit ESC ESC Y wird das Telnet beendet.

9 Filetransfer per FTP

Ist das Softwarepaket nicht in den RTOS-EPROMs enthalten, muß es erst vom Massenspeicher nachgeladen werden. Es stehen ein FTP-Server (ermöglicht den Zugriff auf ein RTOS-System von einem externen Rechner) und ein FTP-Client (ermöglicht den Zugriff auf einen externen Rechner von einem RTOS aus) zur Verfügung.

```
LOAD FTPCLI.SR      der FTP-Client
LOAD FTPSRV.SR      der FTP-Server
```

9.1 FTP-Server

Zum Starten des FTP-Servers muss folgendes Kommado eingegeben werden:

```
FTPSRV [-p=pass_wort_datei] [-t=time_out] [-o=type] [-b]
```

Für einen FTP-Client ist der RTOS-Rechner unter der beim Start des NETIO angegebenen *ip_address* erreichbar.

Der FTP-Server bildet für jede akzeptierte Verbindung eine eigene Subtask.

Mit der Option `-o` kann die Directoriedarstellung umgestellt werden. Das UNIX-Format ist z.B. notwendig, wenn mit einem Web-Browser auf den FTP-Server zugegriffen werden soll. Ist die Ausgabe des Directorys im verwendeten FTP-Client unleserlich oder fehlerhaft, müssen die verschiedenen Einstellungen probiert werden. Folgende Einstellungen sind möglich:

`-o=0`	Directory im RTOS-Format
`-o=1`	Directory im UNIX-Format
`-o=2`	Directory im DOS-Format

Mit der Option `-t` kann die Timeout-Zeit eingestellt werden. Wenn auf einer FTP-Verbindung keine Aktivität feststellbar ist, schließt der Server die Verbindung nach der angegebenen Timeout-Zeit. Sie kann im Minutenraster eingestellt werden, default-Timeout ist 10 Minuten. Wird die Timeout-Zeit auf 0 gesetzt, findet keine Zeitüberwachung statt. Dabei ist zu beachten, dass ein TCP-Port belegt – und damit nicht mehr nutzbar – liegen bleiben kann, wenn der FTP-Client vom Netz getrennt wird, ohne dass er sich abmelden konnte.

Die Option `-b` läßt den Server im Binärmode starten.

9.1.1 FTP-Server Passwort-Datei

Mit der Option `-p` kann dem FTP-Server eine Passwort-Datei zugeteilt werden. In dieser sind Usernamen und Passwörter sowie ggf. Zugriffsbeschränkungen für den jeweiligen User verzeichnet. Es sollte immer der komplette Pfad der Passwortdatei angegeben werden. Die gleiche Passwortdatei kann auch für den Telnet-Server verwendet werden.

Beispiel:

```
FTPSRV -p=/H0/MY_PASSWD
```

9.1.1.1 FTP-Server Klartext Passwortdatei

Die Datei MY_PASSWD habe bei Klartext Passworten folgenden Inhalt:

```
GUEST , GUEST     * User GUEST mit Passwort GUEST
/H2/*             * darf überall auf /H2/ zugreifen und
/H1/C/*           * darf ab /H1/C/... zugreifen
root , Master     * User root mit Passwort Master
/H0/*             * darf  auf  die Platten /H0/
-/H1/*            * /H1/ mit Schreibschutz
/H2/*             * und /H2/ ohne Beschränkung zugreifen
Karl , Xy4bla     * User Karl, Passwort Xy4bla
/H0/Karl/         * darf nur in Directory /H0/Karl/
```

Für jeden User wird seine erste Pfadangabe als Startpfad eingestellt. Endet die Pfadangabe mit einem *****, so darf er auch auf alle Unterverzeichnisse zugreifen. Fehlt der *****, wie bei User Karl, ist der Zugriff nur auf das explizit angegebene Directory möglich. Mit /* kann der Komplettzugriff gestattet werden. Ein – (Minuszeichen) vor dem Pfad gibt an, dass der User auf diesen Dateipfad nur Leserechte hat, er kann dann auf Dateien keinen Put, Rename oder Append durchführen.

Bitte beachten Sie, dass zwischen Nutzernamen und folgendem Komma ein Leerzeichen stehen muss und die Datei keine Leerzeilen enthalten darf!

9.1.1.2 FTP-Server MD5 Passworten

Es besteht ab FTPSRV=6.7 die Möglichkeit in der Passwortdatei das Passwort MD5 zu verschlüsseln, zur Verschlüsselung ist das MD5TOOL notwendig, daß den Schlüssel errechnet. Bei Bedarf können Sie ein entsprechendes Programm bei uns erhalten.

Achtung: das MD5TOOL erzeugt eine anderes MD5 Passwort als andere Tools da hier noch die Erstellungszeit eingerechnet wird .

Die Datei MY_PASSWD habe bei MD5 Passwort folgenden Inhalt:

```
GUEST : Ex6W83924c79d793461accf92cb9b883323a   * User GUEST mit MD5-Passwort für GUEST
/H2/*                                          * darf überall auf /H2/ zugreifen und
/H1/C/*                                        * darf ab /H1/C/... zugreifen
root : Af6S554892f7c15603ee8a3469f99345cf60    * User root mit MD5-Passwort für Master
/H0/*                                          * darf  auf  die Platten /H0/
-/H1/*                                         * /H1/ mit Schreibschutz
/H2/*                                          * und /H2/ ohne Beschränkung zugreifen
Karl : Yx0S8cfb2f2e965c203637aecea60aedd2ca    * User Karl mit MD5 Passwort für Xy4bla
/H0/Karl/                                      * darf nur in Directory /H0/Karl/
```

Für jeden User wird seine erste Pfadangabe als Startpfad eingestellt. Endet die Pfadangabe mit einem *****, so darf er auch auf alle Unterverzeichnisse zugreifen. Fehlt der *****, wie bei User `Karl`, ist der Zugriff nur auf das explizit angegebene Directory möglich. Mit `/*` kann der Komplettzugriff gestattet werden. Ein – (Minuszeichen) vor dem Pfad gibt an, dass der User auf diesen Dateipfad nur Leserechte hat, er kann dann auf Dateien keinen Put, Rename oder Append durchführen.

Bitte beachten Sie, dass zwischen Nutzernamen und folgendem Doppelpunkt ein Leerzeichen stehen muss und die Datei keine Leerzeilen enthalten darf!

9.2 FTP-Client

Starten des FTP-Client:

`FTP [-txxx] [-v`*xx*`] [-c] `*ip_address*`|`*hostname*`[`*batch*` [`*out_redir*`] ]`

ip_address gibt die Adresse des Zielrechners an. Ist auf `/R0/` oder `/H0/ETC/` eine `HOSTS`-Datei vorhanden, kann statt der *ip_address* der *hostname* angegeben werden. Dazu muss er natürlich in der `HOSTS`-Datei eingetragen sein.

Wurde ein *batch* angegeben, so werden die FTP-Kommandos aus dieser Datei abgearbeitet. Mit *out_redir* kann eine Datei angegeben werden, in die alle Ausgaben des FTP umgelenkt werden.

Mit der Option `-t` kann die Timeout-Zeit eingestellt werden. Wenn auf einer FTP-Verbindung keine Aktivität feststellbar ist, schließt der Client die Verbindung nach der angegebenen Timeout-Zeit. Sie kann im Sekundenraster eingestellt werden. Wird die Timeout-Zeit auf 0 gesetzt, findet keine Zeitüberwachung statt (default).

Mit der `-v` Option kann die Anzahl der Versuche bei der Verbindungseröffnung eingestellt werden. Der Default-Wert ist 10.

Ist der `-c` Parameter mit angegeben, so wird die TCP-Verbindung im Fehlerfall automatisch geschlossen und auch die Batchabarbeitung sofort abgebrochen.

Bei Beendigung des FTP wird der Returnstatus gesetzt, damit ist es möglich zu überprüfen, ob der Batch-Betrieb fehlerfrei gelaufen ist.

Die Standard-Parameter beim Start des FTP sind:

- ASCII-Transfermode
- NONINTERACTIVE
- VERBOSE angeschaltet
- DEBUG ausgeschaltet
- PAssive Mode ausgeschaltet

9.2.1 Aufbau des Batchfiles

Die Batch-Datei zur Steuerung des FTP habe folgenden Aufbau:

```
Kroll           * der Username
Hallo           * ggf. Passwort
CD /H0/KROLL    * Setzen des Directory (remote)
LCD /H1/        * Setzen des Directory (local)
MGET *          * hole alle Files vom Remote
QUIT            * Beenden der Session
^D              * CTRL D als Endekennzeichen
```

Die Kommandos der Batch-Datei werden in der Reihenfolge ihres Auftretens abgearbeitet. Bei der Beendigung des FTP-Client wird der Returnstatus gesetzt. Damit kann geprüft werden, ob **alle** Kommandos erfolgreich abgearbeitet wurden. Falls ein Kommando fehlerhaft bearbeitet wurde, so wird mit dem nächsten Kommando im Batchfile fortgefahren, der Returnstatus wird auf "fehlerhaft" gesetzt.

9.2.2 FTP-Client-Kommandos

Alle Kommandos können in Groß- oder Kleinschreibung eingegeben werden. Für die meisten Kommandos sind 2 oder 3 Buchstaben lange Abkürzungen zulässig, eine Liste befindet sich am Ende des Kapitels.

9.2.2.1 Verbindungsaufbau und -abbruch

Kommando	Parameter	Bedeutung
`QUIT` `BYE`		Beenden der Sitzung (terminiert den FTP-Client)
`CLOSE`		Beenden der Verbindung (schließt Verbindung)
`OPEN`	*`ip_addr`*	Öffnen einer Verbindung (meist implizit beim Start) Die *`ip_addr`* muß in der bekannten Oktett Schreibweise eingegeben werden: z.B. `192.168.10.3`
`USER`	*`user_name`*	Sie melden sich unter dem neuen Namen *`user_name`* an.

9.2.2.2 Übertragungsarten

Kommando	Parameter	Bedeutung
`ASCII`		Dateitransfer im ASCII-Mode ASCII-Übertragung unter FTP bedeutet, daß die Dateien als Textdateien über das Netz übertragen werden. Die Zeilenende-Kennung ist CR/LF. Unter RTOS-UH wird in allen empfangenen Dateien das CR/LF in ein CR umgewandelt. Bei den gesendeten Dateien wird ein CR am Zeilenende in ein CR/LF geändert.
`BINARY`		Dateitransfer im BINARY-Mode Binary-Übertragung unter FTP bedeutet, daß alle Daten ohne Veränderung übertragen werden. Vorsicht: Werden Textdateinen im Binary-Mode übertragen, können sie u.U. auf einem anderen System nicht mehr bearbeitet werden.

9.2.2.3 Directory-Behandlung auf dem Remote-Rechner

Kommando	Parameter	Bedeutung
`DIR`	*`path_list`*	Anzeigen des Directorys des Remote Rechners. Es wird das komplette Directory mit Datum, Uhrzeit und Größe ausgegeben.

Beispiele:

`DIR`	Es werden alle Dateien auf dem aktuell eingestellten Directory ausgegeben.
`DIR K*`	Auf dem aktuell eingestellten Directory werden alle Dateien, bei denen der Name mit `K` beginnt, ausgegeben.
`DIR /user/iep/*.*`	Auf dem Directory `/user/iep` werden alle Dateien mit Extension ausgegeben.
`DIR *.c`	Auf dem aktuell eingestellten Directory werden alle Dateien mit der Extension `*.c` ausgegeben.

Kommando	Parameter	Bedeutung
`LS`	*`path_list`*	Anzeigen des Directorys des Remote Rechners. Es wird das komplette Directory ohne Datum, Uhrzeit und Größe ausgegeben.
`CD`	*`path_list`*	Das Working-Directory wird auf *`path_list`* gesetzt.

Beispiele:

`CD KR`	Es wird relativ zum aktuell eingestellten Verzeichnis in das Unterverzeichnis **KR** gewechselt.
`CD /user/iep`	Es wird unabhängig vom eingestellten Verzeichnis nach **`/user/iep`** gewechselt.

Kommando	Parameter	Bedeutung
`PWD`		Anzeige des eingestellten Directory-Pfades.
`MKDIR`	*`path_list`*	Es wird ein neues Directory angelegt.
`RMDIR`	*`path_list`*	Es wird das mit *`path_list`* bezeichnete Verzeichnis gelöscht

9.2.2.4 Directory-Behandlung auf dem Localen-Rechner

Wird vor die Befehle zur Directoryeinstellung auf dem Remote-Rechner der Buchstabe **`L`** (=local) gesetzt, wirken die Befehle auf den lokalen Rechner. So wird aus **`DIR`** ein **`LDIR`**, um das eigene Directory anzuzeigen.

9.2.2.5 Dateitransfer einzelner Dateien

Kommando	Parameter	Bedeutung
`GET` `BGET`	*`rem_datei`* [*`loc_name`*]	Es wird eine Datei vom Remote- zum lokalen Rechner übertragen. Wurde kein *`loc_name`* angegeben, wird der bei *`rem_datei`* angegebene Datei-Name als lokale Name verwendet. Je nach eingestellter Übertragungsart wird die Datei nicht geändert (binary) oder die Zeilenende-Kennung wird angepaßt. Der Befehl `BGET` überträgt immer im Binary-Mode.

Beispiele:

`GET test` Es wird die Datei `test` vom Remote-Rechner auf den lokalen Rechner übertragen. Der Name `test` wird beibehalten.

`GET /user/a bb` Es wird die Datei `a` vom Verzeichnis `/user/` des Remote-Rechners auf den lokalen Rechner übertragen. Dort wird sie unter dem Namen `bb` abgespeichert.

Kommando	Parameter	Bedeutung
`PUT` `BPUT`	*`loc_datei`* [*`rem_name`*]	Es wird eine Datei vom Lokalen- zum Remote-Rechner übertragen. Wurde kein *`rem_name`* angegeben, wird der bei *`loc_datei`* angegebene Datei-Name als Remote-Name verwendet. Je nach eingestellter Übertragungsart wird die Datei nicht geändert (binary) oder die Zeilenende-Kennung wird angepaßt. Der Befehl `BPUT` überträgt immer im Binary-Mode.

Beispiele:

`PUT test` Es wird die Datei `test` vom lokalen Rechner auf den Remote-Rechner übertragen. Der Name `test` wird beibehalten.

`PUT /user/a bb` Es wird die Datei `a` vom Verzeichnis `/user/` des lokalen Rechners auf den Remote-Rechner übertragen. Dort wird sie unter dem Namen `bb` abgespeichert.

Kommando	Parameter	Bedeutung
`APPEND` `APP`	*`loc_datei`* [*`rem_name`*]	Es wird eine Datei vom lokalen zum Remote-Rechner übertragen. Falls die Datei auf dem Remote-Rechner existiert, wird sie ans Ende angehängt. Wurde kein *`rem_name`* angegeben, wird der bei *`loc_datei`* angegebene Datei-Name als lokaler Name verwendet. Je nach eingestellter Übertragungsart wird die Datei nicht geändert (binary) oder die Zeilenende-Kennung wird angepaßt.

Beispiele:

`APPEND test` Es wird die Datei `test` vom lokalen Rechner auf den Remote-Rechner übertragen. Dort wird sie an eine Datei `test` angehängt.

`APP /user/a bb` Es wird die Datei `a` vom Verzeichnis `/user/` des lokalen Rechners auf den Remote-Rechner übertragen. Dort wird sie an die Datei `bb` angehängt.

Kommando	Parameter	Bedeutung
`DELETE` `RM`	*`rem_datei`*	Es wird die Datei *`rem_datei`* auf dem Remote-Rechner gelöscht.

Beispiele:

`DELETE test` Es wird die Datei `test` auf dem Remote-Rechner gelöscht.

`RM /user/a` Es wird die Datei `a` auf dem Verzeichnis `user` des Remote-Rechners gelöscht.

Kommando	Parameter	Bedeutung
`RENAME`	*`old_datei new_datei`*	Es wird die Datei *`old_datei`* auf dem Remote-Rechner in die Datei *`new_datei`* umbenannt.

Beispiele:

`RENAME test bla` Die Datei `test` auf dem Remote-Rechner wird in `bla` umbenannt.

9.2.2.6 Dateitransfer mehrerer Dateien

Kommando	Parameter	Bedeutung
`MGET`	*`file_mask`*	Alle Dateien, auf die die *`file_mask`* zutrifft, werden vom Remote-Rechner auf den lokalen Rechner übertragen. Die Dateinamen werden beibehalten. Es wird die eingestellte Übertragungsart verwendet. Als Platzhalter für beliebige Zeichen ist der ***** zulässig.
`MPUT`	*`file_mask`*	Alle Dateien, auf die die *`file_mask`* zutrifft, werden vom lokalen Rechner auf den Remote-Rechner übertragen. Die Dateinamen werden beibehalten. Es wird die eingestellte Übertragungsart verwendet. Als Platzhalter für beliebige Zeichen ist der ***** zulässig.
`MDELETE`	*`file_mask`*	Alle Dateien, auf die die *`file_mask`* zutrifft, werden auf dem Remote-Rechner gelöscht.
`INTERACTIVE`		Für jede Datei, die bei den obigen Kommandos bearbeitet wird, ist eine Bestätigung vom Anwender einzugeben.
`NONINTERACTIVE`		Das Kommado wird auf alle Dateien angewendet, ohne das der Anwender eingreifen muß oder kann.
`PROMPT`		Toggelt das `INTERACTIVE` Verhalten.

9.2.2.7 Beobachtung der Übertragung

Kommando	Parameter	Bedeutung
VERBOSE		Es werden die übertragenen Bytes, die Übertragungszeit und die Übertragungsrate ausgegeben. Jede Eingabe von VERBOSE toggelt den Mode.
DEBUG		Alle internen Kommandos werden ausgegeben. Jede Eingabe von DEBUG toggelt den Mode.
HASH		Für jeden übertragenen 1024er-Datenblock wird ein # ausgegeben. Jede Eingabe von HASH toggelt den Mode.

9.2.2.8 Allgemeine Befehle

Kommando	Parameter	Bedeutung
?		Befehlsübersicht
!		Das FTP auf dem eigenen Rechner wird suspendiert. Es wird die Meldung >>FTP/*xxxx*-Client suspended ausgegeben. Damit ist wieder die Kommando-Shell des RTOS-UH erreichbar. Das FTP kann mit c FTP/*xxxx* fortgesetzt werden.
BELL		Nach jedem Kommando wird ein BELL (0x07) ausgegeben.
CLRerr		Der Fehlerzähler wird gelöscht
HELP	[*Kommando*]	Wird *Kommando* angegeben, so wird eine Erklärung zu dem angegebenen Kommando ausgegeben, ansonsten eine Liste der Kommandos mit Erklärung.
REMOTEHELP	[*Kommando*]	Es wird die Hilfe des Remote-FTP ausgegeben.
QUOTE	*rKommando*	Das rKommando wird an den Remote-Server weitergegeben. Dort wird es ausgeführt.
STATUS		Der Status des FTP wird ausgegeben.
PAssive		Der Client wird auf den Passive Mode umgeschaltet. Jede Eingabe von PASSIVE toggelt die Betriebsart. Der Passive-Mode wird z.B. benötigt, wenn der FTP-Client über eine Firewall auf den FTP-Server zugreifen will. Für die Datenübertragung teilt der Server dem Client ein Port mit, die Verbindung zu diesem Port wird dann vom Client geöffnet.

Kommando	Parameter	Bedeutung
BATCH	*batch_datei* [*output_redir*]	Es wird ein Batchprozess gestartet. Die *batch_datei* enthält eine Reihe von FTP-Kommandos. Der Aufbau der Batch-Datei ist in Kapitel 9.2.1 beschrieben. Die Ausgaben des FTP-Clients können in die Datei *output_redir* umgeleitet werden.

Beispiele:

`BATCH /H0/doit /H0/prot` Die Datei `/H0/doit` wird zeilenweise gelesen und abgearbeitet. Alle Ausgaben des FTP's werden in die Datei `/H0/prot` geschrieben.

9.2.3 Kurzübersicht FTP-Kommandos

Für den Aufruf der Befehle reicht die Angabe der in dieser Liste angegebenen GROSS-Buchstaben als Kurzform.

Kommando	Parameter	Bedeutung
?		Befehlsübersicht
!		Suspendierung des eigenen FTP-Client
APPend	*loc_datei* [*rem_name*]	*loc_datei* an Remote-Datei anhängen
ASCii		Umschalten auf ASCII-Mode
BATch	*batch_datei* [*redir*]	Batchprozess starten
BELl		Bei Kommandoende Bell ausgeben
BGet	*rem_datei* [*loc_name*]	Holen einer Datei im Binary-Mode
BINary		Umschalten auf Binary-Mode
BPut	*loc_datei* [*rem_name*]	Senden einer Datei im Binary-Mode
BYE		Beenden der Sitzung
CD	*path_list*	Wechseln des Remote-Verzeichnisses
CLose		Schliessen der Verbindung
CLRerr		Löschen des Fehlerzählers
DEBug		Debug-Mode toggeln
DELete	*rem_datei*	Löschen der Remote-Datei
DIR	*path_list*	Ausgabe des Remote-Directorys (Langform)
GET	*rem_datei* [*loc_name*]	Holen einer Datei
HAsh		Übertragung anzeigen
HElp	[*Kommando*]	Lokale Hilfe
INteractive		Benutzer-Abfrage Flag setzen
LCD	*path_list*	Wechseln des lokalen Directorys
LDIR	*path_list*	Anzeige des lokalen Directorys (Langform)
LLS	*path_list*	Anzeige des lokalen Directorys (Kurzform)
LPWD		Ausgabe des lokalen Pfades
LS	*path_list*	Anzeige des Remote-Directorys (Kurzform)
MDElete	*file_mask*	Löschen von Dateien
MGet	*file_mask*	Holen von Dateien
MKDir	*path_list*	Anlegen eines Directorys
MPut	*file_mask*	Senden von Dateien
NONINteractive		Benutzer-Abfrage Flag löschen
OPen	*ip_addr*	Öffnen einer Verbindung
PAssive		Passive Mode toggeln
PROMpt		Benutzer-Abfrage Flag toggeln
PUT	*loc_datei* [*rem_name*]	Senden einer Datei
PWD		Ausgabe des Remote-Pfades
QUIT		Beenden des FTP-Client
QUOTE	*remote_command*	Absetzen von Sonderkommandos

Kommando	Parameter	Bedeutung
REMotehelp		Hilfe des Remote-FTP
REName	*old_name new_name*	Umbennen einer Datei
RM	*rem_datei*	Löschen einer Datei
RMDir	*rem_directory*	Löschen eines Verzeichnisses
STatus		Status des eigenen FTP's
USER	*user_name*	Nutzerkennung ändern
VERBose		Übertragungswerte anzeigen

10 Die Netzwerk-Programmierschnittstelle unter RTOS-UH

Für den Aufbau von eigenen Kommunikationsverbindungen stehen unter RTOS-UH TCP (Transmission Control Protocol) und UDP (User Datagramm Protocol) zur Verfügung. Beiden ist gemeinsam, dass eine Punkt zu Punkt-Verbindung aufgebaut wird. Die Verbindung ist mit 4 Parametern vollständig beschrieben:

```
Quell-IP-Adresse, Quell-Port, Ziel-IP-Adresse, Ziel-Port
```

Die IP-Adresse beschreibt eindeutig den Rechner, mit der Port-Nummer wird der Prozeß auf dem jeweiligen Rechner adressiert.

10.1 Einsatz von TCP oder UDP

Der Unterschied zwischen TCP und UDP besteht in der Sicherheit der Verbindung. TCP garantiert, dass keine Daten verloren gehen oder in der Reihenfolge vertauscht werden. Auf UDP-Verbindungen können sowohl Daten verloren gehen als auch in der Reihenfolge vertauscht werden. Es liegt in Ihrer Hand als Programmierer der Anwendung dafür zu sorgen, dass Ihre Anwendung damit umgehen kann. Der Vorteil von UDP ist der geringere Protokolloverhead, damit u.U. eine höhere Übertragungsgeschwindigkeit sowie eine geringere Prozessorbelastung.

10.2 Transmission Control Protocol (TCP)

10.2.1 Aufgaben des TCP

Über TCP wird eine Punkt zu Punkt Verbindung zwischen zwei Kommunikationspartnern aufgebaut. Sie ist durch vier Parameter eindeutig festgelegt:

```
Quell-Adresse, Ziel-Adresse, Quell-Port, Ziel-Port
```

Die Adresse legt den angesprochenen Rechner fest, das Port bestimmt die Anwendung auf dem Rechner. Die Adresse ist eine eindeutige IP-Adresse, die Portnummer sollte für eigene Anwendungen im Bereich zwischen 4096 und 32767 liegen. Unter 4096 liegen Ports, die für bestimmte Anwendungen reserviert sind. Die Anzahl der maximal gleichzeitig geöffneten Ports ist nur durch den zur Verfügung stehenden Speicher begrenzt.

Die Anwenderdaten werden in TCP-Paketen übertragen. Die Länge pro Paket darf zwischen 1 und 1436 Byte liegen.

10.2.1.1 Aufbau eines TCP-Paketes

Für die Datenübertragung werden TCP-Pakete genutzt, diese haben einen festgelegten Aufbau. Sie als Anwender brauchen sich um diesen Aufbau nicht kümmern, der TCP-Stack erzeugt entsprechende Pakete.

Ein Paket besteht aus dem Header- und dem Data-Teil. Im Header sind die für die Netzwerkverwaltung benötigten Daten abgelegt :

```
Hardware-Ziel-Adresse   /* vom TCP-Stack erzeugt */
Hardware-Quell-Adresse  /*            "          */
Pakettype               /*            "          */
IP-Header               /*            "          */
TCP-Header              /* über TCP-Funktion     */
Userdaten               /* Anwenderdaten         */
```

Im Data-Teil eines Paketes sind die vom Anwender zu übermittelnden Daten abgelegt. Pro Paket können minimal 1 Byte und maximal 1436 Byte übertragen werden. Pakete mit anderen Längen werden nicht akzeptiert und mit einer Fehlermeldung vom TCP-Stack abgewiesen.

10.2.2 Die TCP-Programmierschnittstelle

Die folgende Beschreibung ermöglicht es Ihnen eigene TCP-Verbindungen zu programmieren. Als Programmiersprache werden PEARL und CREST-C unterstützt. Für beide Programmiersprachen stehen identische Funktionen zur Verfügung, deshalb unterscheidet die Beschreibung auch nur die Datenstrukturen. Bei PEARL ist der Präfix „TCP_“ und bei CREST-C ist der Präfix „tcp_“, ansonsten sind die Funktionsnamen identisch.

Achtung : Bitte bei PEARL-Programmen den F_NAME mit TOCHAR(0) oder TOCHAR(128) abschliessen.

10.2.2.1 TCP-Library

Um die Programmierschnittstelle für die Sprache PEARL zu nutzen muss die PEARL-Funktionsbibliothek NETLIB.SR geladen sein. Für CREST-C sind alle Programme mit der NETLIB.LIB zu linken.

10.2.2.2 TCP Socket Struktur

Die Schnittstelle zum TCP-Stack wird mit der TCP-Socket-Struktur realisiert:

PEARL

```
TYPE TCP_SOCKET STRUCT
(/
  DEST_ADDR   BIT(32)
  SOURCE_PORT FIXED(15)
  DEST_PORT   FIXED(15)
  TIME_OUT    FIXED(31)
  F_NAME      CHAR(32)
  SOURCE_ADDR BIT(32)
  PUSH_FLAG   FIXED(15)
  URG_FLAG    FIXED(15)
  SND_WND     FIXED(31)
  RCV_WND     FIXED(31)
  T_STATE     FIXED(15)
/);
```

CREST-C

```
struct tcp_socket
{
  u_long    dest_addr  ;  /*destination ip-adr*/
  u_short   source_port;  /*source port       */
  u_short   dest_port  ;  /*destination port  */
  int       time_out   ;  /*timeout len       */
  char      f_name[32] ;  /*Filename des Kanal*/
  u_long    source_addr;  /*source ip-adr     */
  u_short   push_flag  ;  /*push flag         */
  u_short   urg_flag   ;  /*urgent flag       */
  int       snd_wnd    ;  /*send window       */
  int       rcv_wnd    ;  /*receive window    */
  u_short   t_state    ;  /*state of TCP      */
};
```

Kommentar

Zum Aufbau einer eindeutigen Verbindung müssen folgende Bedingungen eingehalten werden:

Quelle (Sender)		Ziel (Receiver)
`Source_Port`	=	`Destination_Port`
`Destination_Port`	=	`Source_Port`
`Destination_IP`	=	`eigene IP`

Soll ein Server realisiert werden, der einen Verbindungswunsch von einem beliebigen Rechner und /oder Port entgegen nehmen kann, so dürfen beim `TCP_passive_open` `Destination_IP` und/oder `Destination_Port` auf 0 gesetzt werden.

10.3 TCP Funktionen

Für das TCP stehen die folgenden Funktionen zur Verfügung:

Funktion	Bedeutung
`TCP_active_open`	Aufbau einer TCP-Verbindung (Client)
`TCP_passive_open`	Auf den Aufbau eine Verbindung warten (Server)
`TCP_status`	Status einer Verbindung abfragen
`TCP_close`	Schliessen einer Verbindung und löschen des Socket
`TCP_abort`	Abbrechen einer Verbindung und löschen des Socket
`TCP_send`	Senden von Daten über die Verbindung
`TCP_receive`	Empfangen von Daten
`TCP_Nreceive`	Empfangen von Daten mit Begrenzung der Länge
`TCP_cleanup`	Beenden **aller** Verbindungen!!! Reset des TCP-Stack!
`TCP_set_timeout`	Setzen des Empfangstimeout
`TCP_reset`	Schliessen der Verbindung, an die Gegenseite wird ein RESET geschickt. Der Socket wird nicht gelöscht!
`TCP_free_socket`	Der Speicher des Sockets wird freigegeben, darf nur in Kombination mit TCP_reset genutzt werden !
`TCP_passive_bind`	Einrichten der Verbindung im lokalen Stack
`TCP_wait_listen`	Warten auf komplette Verbindungseröffnung
`TCP_wait_accept`	Längeres Warten auf komplette Verbindungseröffnung
`TCP_Nsend`	Senden von Daten über die Verbindung, auch unvollständig als Teil des angegebenen Puffers

Die Verbindung des durch `TCP_`*xx*`_open` geöffneten Ports zum Programm erfolgt über die Struktur `tcp_socket`. In dieser Struktur werden interne Zustände des Protokollstacks vermerkt, deshalb darf der Socket zwischen einem `TCP_`*xx*`_open` und einem `TCP_close` nicht verändert werden! Weiterhin sind eindeutige Filenamen beim `TCP_`*xx*`_open` zu verwenden! Mit dem Aufruf des `TCP_`*xx*`_open` wird auch der Speicher für die Struktur `tcp_socket` besorgt. Dieser Speicher wird mit dem `TCP_close` oder `TCP_abort` wieder an das Betriebssystem zurück gegeben, danach und wärendessen darf keine andere `TCP_funktion` genutzt werden, es kann sonst zu beliebigen Fehlfunktionen des gesamten Systems kommen.

10.3.1 TCP_active_open

Die Funktion `TCP_active_open` erzeugt einen Socket für das angegebene Port/IP-Adresse und versucht die Verbindung zum Zielrechner zu öffnen. Dazu muss auf dem Zielrechner auf eine Verbindungseröffnung gewartet werden (z.B. mit einem `TCP_passive_open`). Kann keine Verbindung aufgebaut werden, so wird der `TCP_active_open` mit der entsprechenden Fehlermeldung beendet.

Die Funktion `TCP_active_open` erwartet folgende Parameter:

Parameter	PEARL-Type	C-Type	Bedeutung
Dest_addr	`FIXED(31)`	`u_long`	IP-Adresse des Zielrechners
D_Port	`FIXED(15)`	`u_short`	Destination Port auf dem Zielrechner
S_Port	`FIXED(15)`	`u_short`	Source Port auf dem eigenen Rechner
Timeout	`FIXED(31)`	`int`	max Wartezeit beim Verbindungsaufbau
F_Name	`CHAR(1)`	`char *`	Filenamen der Verbindung

Die Funktion `TCP_active_open` gibt einen Pointer auf den erzeugten Socket zurück.

Besonderheit in PEARL:

Bei einem PEARL-Programm wird der Rückgabepointer in einer Struktur versteckt. Der Filename wird mit `F_Name.CHAR(1)` übergeben.

Rückgabe-Struktur:

```
TYPE TCP_STRUCT [ pSocket REF TCP_SOCKET ] ;
```

Wird Timeout auf 0 gesetzt, findet keine Zeitüberwachung statt. Die Zeit wird in Millisekunden angegeben. Da ggf. bis zu 10 Mal versucht wird die Verbindung herzustellen, ist die TimeOut-Zeit max. 10 mal so lang wie angegeben.

10.3.1.1 Beispiel TCP_active_open

PEARL-Programm:

```
...
DCL conn   TCP_STRUCT ;
DCL f_name CHAR(24)   ;
f_name = 'My_Channel'><TOCHAR(0) ;
...
conn.pSocket = NIL ; /* vorinitialiseren */
conn = TCP_active_open( TOFIXED('C1010101'B4), 4096, 4097, 0(31),
                        f_name.CHAR(1) );
IF conn.pSocket ISNT NIL THEN
   /* Verbindung erfolgreich geoeffnet     */
   /* Schreiben und/oder Lesen von Daten   */
   ...
   errcode = TCP_close( conn ) ; /* Verbindung schliessen */
   ...
ELSE
```

```
      /* keine Verbindung geoeffnet       */
   FIN;
```

C-Programm:

```
   ...
   tcp_socket *conn       ;
   char        f_name[24] = "My_Channel" ;
   ...
   conn = tcp_active_open( 0xC1010101L, 4096, 4097, 0, f_name );
   if ( conn != NULL )
   {
      /* Verbindung erfolgreich geoeffnet     */
      /* Schreiben und/oder Lesen von Daten   */
      ...
      errcode = tcp_close( conn ) ; /* Verbindung schliessen */
      ...
   }
   else
   {
      /* keine Verbindung geoeffnet       */
   }
```

Wird über die obige Verbindung gelesen, werden nur Pakete des Absenders mit der IP-Adresse `'C1010101'B4` (entspricht `193.1.1.1`) akzeptiert, wenn weiterhin bei diesem als Source-Port 4097 und als Destination-Port 4096 eingetragen ist. Alle Pakete, die diese Bedingungen nicht erfüllen, werden nicht an das Programm durchgereicht.

Werden über die obige Verbindung Daten gesendet, wird die Station mit der IP-Adresse `'C1010101'B4` (entspricht `193.1.1.1`) angesprochen, dabei ist als Source-Port 4096 und Destination-Port 4097 eingetragen.

10.3.2 TCP_passive_open

Die Funktion `TCP_passive_open` erzeugt einen Socket für das angegebene Port/IP-Adresse und wartet auf die Eröffnung einer Verbindung. Die Funktion wartet bis entweder eine Gegenseite die Verbindung erfolgreich eröffnet hat oder das angegebene Timeout abgelaufen ist.

Die Funktion `TCP_passive_open` erwartet folgende Parameter:

Parameter	PEARL-Type	C-Type	Bedeutung
Dest_addr	`FIXED(31)`	`u_long`	IP-Adresse des Zielrechners
D_Port	`FIXED(15)`	`u_short`	Destination Port auf dem Zielrechner
S_Port	`FIXED(15)`	`u_short`	Source Port auf dem eigenen Rechner
Timeout	`FIXED(31)`	`int`	max Wartezeit beim Verbindungsaufbau
F_Name	`CHAR(1)`	`char *`	Filenamen der Verbindung

Die Funktion `TCP_passive_open` gibt einen Pointer auf den erzeugten Socket zurück.

Besonderheit in PEARL:

Bei einem PEARL-Programm wird der Rückgabepointer in einer Struktur versteckt. Der Filename wird mit `F_Name.CHAR(1)` übergeben.

Rückgabe-Struktur:

```
TYPE TCP_STRUCT [ pSocket REF TCP_SOCKET ] ;
```

Wird Timeout auf 0 gesetzt, findet keine Zeitüberwachung statt. Die Zeit wird in Millisekunden angegeben. Da ggf. 2 Mal versucht wird die Verbindung herzustellen, ist die TimeOut-Zeit max. doppelt so lang wie angegeben.

10.3.2.1 Beispiele TCP_passive_open

PEARL-Programm:

```
...
DCL conn   TCP_STRUCT ;
DCL f_name CHAR(24)   ;
f_name = 'My_Channel'><TOCHAR(0) ;
...
conn.pSocket = NIL ; /* vorinitialiseren */
conn = TCP_passive_open( TOFIXED('C1010101'B4), 4096, 4097, 0(31),
                          f_name.CHAR(1)
                        );
IF conn.pSocket ISNT NIL THEN
   /* Verbindung erfolgreich geoeffnet     */
   /* Schreiben und/oder Lesen von Daten   */
   ...
   errcode = TCP_close( conn ) ; /* Verbindung schliessen */
   ...
ELSE
   /* keine Verbindung geoeffnet       */
FIN;
```

C-Programm:

```
...
tcp_socket *conn ;
char        f_name[24] = "My_Channel" ;
...
conn = tcp_passive_open( 0xC1010101L, 4096, 4097, 0L, f_name );

if ( conn != NULL )
{
   /* Verbindung erfolgreich geoeffnet     */
   /* Schreiben und/oder Lesen von Daten   */
   ...
   errcode = tcp_close( conn ) ; /* Verbindung schliessen */
```

```
   ...
}
else
{
   /* keine Verbindung geoeffnet       */
}
```

Wird über die obige Verbindung gelesen, werden nur Pakete des Absenders mit der IP-Adresse `'C1010101'B4` (entspricht `193.1.1.1`) akzeptiert, wenn weiterhin bei diesem als Source-Port 4097 und als Destination-Port 4096 eingetragen ist. Alle Pakete, die diese Bedingungen nicht erfüllen, werden nicht an das Programm durchgereicht.

Werden über die obige Verbindung Daten gesendet, wird die Station mit der IP-Adresse `'C1010101'B4` (entspricht `193.1.1.1`) angesprochen, dabei ist als Source-Port 4096 und Destination-Port 4097 eingetragen.

10.3.3 TCP_passive_bind, TCP_wait_listen, TCP_wait_accept

Die Funktion `TCP_passive_bind` erzeugt einen Socket für die angegebene Port/IP-Adresse. Der Rückgabewert ist >0 falls der Socket angelegt werden konnte.

Die Funktion `TCP_wait_listen` wartet auf die Verbindungseröffnung (ein SYN der Gegenseite) Mit der beim passive_bind eingestellten Timeout-Zeit wird die Wait-Variable überprüft ob sie noch auf dem Wert=1 steht, falls nicht wird das Warten abgebrochen werden soll.

In diesem Fall kann mit `TCP_wait_accept` weiter auf die erfolgreiche Verbindungseröffnung gewartet werden. Das Timeout ist einstellbar, ggf. darf `TCP_wait_accept` auch mehrfach hintereinander aufgerufen werden.

Der Vorteil dieser Funktionen liegt in der optimalen Server-Programmierung: Da die Funktion `TCP_wait_listen` schon nach dem Empfang des ersten SYN-Paketes von der Gegenseite zum Anwenderprogramm zurückkommt, kann ein paraller Server für das gleiche lokale Port aufgesetzt werden. Damit ist eine bessere Reaktionzeit auf externe Verbindungseröffnungen möglich, auch kann der Server durch das Schicken von SYN-Paketen nicht blockiert werden.

Die Funktion `TCP_passive_bind` erwartet folgende Parameter:

Parameter	PEARL-Type	C-Type	Bedeutung
Dest_addr	`FIXED(31)`	`u_long`	IP-Adresse des Zielrechners
D_Port	`FIXED(15)`	`u_short`	Destination Port auf dem Zielrechner
S_Port	`FIXED(15)`	`u_short`	Source Port auf dem eigenen Rechner
Timeout	`FIXED(31)`	`int`	max. Wartezeit beim Verbindungsaufbau
F_Name	`CHAR(1)`	`char *`	Filenamen der Verbindung

Die Funktion `TCP_wait_listen` erwartet folgende Parameter:

Parameter	PEARL-Type	C-Type	Bedeutung

Sock
Wait
	`TCP_STRUCT`
	`FIXED(31)`
	`IDENT`
		`Tcp_socket*`
		`Int *`
			Socket der Verbindung
			Pointer auf Variable „Wait“, hiermit kann von extern der Wartevorgang abgebrochen werden, falls der Programm beendet werden soll, damit der Socket wieder freigegeben wird.

Die Funktion `TCP_wait_accept` erwartet folgende Parameter:

Parameter	PEARL-Type	C-Type	Bedeutung
sock	`TCP_STRUCT`	`Tcp_socket*`	Socket der Verbindung
Timeout	`FIXED(31)`	`int`	max Wartezeit für Verbindungsaufbau

Die Funktion `TCP_passive_bind` gibt einen Pointer auf den erzeugten Socket zurück.

Besonderheit in PEARL:

Bei einem PEARL-Programm wird der Rückgabepointer in einer Struktur versteckt. Der Filename wird mit `F_Name.CHAR(1)` übergeben.

Rückgabe-Struktur:

```
TYPE TCP_STRUCT [ pSocket REF TCP_SOCKET ] ;
```

Wird das Timeout auf `0` gesetzt, findet keine Zeitüberwachung statt. Die Zeit wird in Millisekunden angegeben. Da ggf. 2 Mal versucht wird, die Verbindung herzustellen, ist die Timeout-Zeit max. doppelt so lang wie angegeben.

10.3.3.1 Beispiele TCP_passive_bind ,TCP_wait_listen,TCP_wait_accept

PEARL-Programm:

```
    ...
    DCL conn   TCP_STRUCT ;
    DCL f_name CHAR(24)   INIT(‘ ‘) ;
    DCL wait_p  FIXED(31) INIT(1(31));
...
    conn.pSocket = NIL ; /* vorinitialiseren */
    WHILE conn.pSocket IS NIL REPEAT ;
       CONVERT ‘MY_SERV_7000‘,TOBIT(next),TOCHAR(0) TO f_name
                 BY A,B4(4),A;
       next = next + 1 ;
       conn = TCP_passive_bind( TOFIXED('00000000'B4), 0, 7000, 0(31),
                              f_name.CHAR(1) );
```

```
END ;
/* Socket eingerichtet  */
errcode = TCP_wait_listen( conn , wait_p ) ; /* SYN von Gegenseite */
/* jetzt kann ein paraller Server aufgesetzt werden */
do_my_next_server( 7000 ) ; /* für gleiches Port */
WHILE errcode GE 0 AND conn.pSocket.t_STATE EQ 3 REPEAT ;
  /* warte bis Verbindung Established */
  errcode = TCP_wait_accept( conn ,2000(31)) ;
END ;
IF errcode GE 0 THEN
   /* jetzt steht Verbindung */
   /* Schreiben und/oder Lesen von Daten   */
   ...
ELSE
   /* Verbindung gescheitert  */
FIN;
errcode = TCP_close( conn ) ; /* Verbindung schliessen */
```

C-Programm:

```
...
tcp_socket *conn ;
char        f_name[24] ;
static short next_srv  ;
int   wait = 1;
...
conn = NULL ;
while( conn == NULL )
{
   sprintf( f_name ,“my_serv_7000%04X“ , next_srv++ ) ;
   conn = tcp_passive_bind( 0 , 0, 7000, 0L, f_name );
}
/* Socket eingerichtet  */
errcode = tcp_wait_listen( conn , &wait ) ; /* SYN von Gegenseite */
/* jetzt kann ein paraller Server aufgesetzt werden */
do_my_next_server( 7000 ) ; /* für gleiches Port */
while( errcode >= 0 && conn.t_state == 3 )
{
    /* Weiter warten auf Established */
    errcode = tcp_wait_accept( conn , 2000) ;
}
if ( errcode >= 0 )
{
      /* jetzt steht Verbindung */
      /* Schreiben und/oder Lesen von Daten   */
     ...
```

```
}
else
{
   /* Verbindung gescheitert      */
}
errcode = tcp_close( conn ) ; /* Verbindung schliessen */
```

Wird über die obige Verbindung gelesen, werden Pakete aller Absenders akzeptiert, in die das Destination-Port 7000 eingetragen ist. Alle Pakete, die diese Bedingungen nicht erfüllen, werden nicht an das Programm durchgereicht.

10.3.4 TCP_close

Die Funktion `TCP_close` schließt die Verbindung geordnet, d.h. beide beteiligten Stationen beenden die Verbindung. Falls die Gegenstation den Verbindungsabbau nicht vollständig durchführt, bleibt in der Protokollsoftware das Port belegt, bis das globale Time-Out abgelaufen ist.

Wenn die Funktion fehlerfrei durchlaufen wurde, ist danach der Socket geschlossen, d.h. in der TCP-Socket-Struktur ist der Socketpointer gelöscht.

Bitte beachten Sie, dass auch der Speicher, den die TCP-Socket-Struktur belegt hat, an das Betriebssystem zurück gegeben wurde! Es darf danach kein weiterer `TCP_`*`funktionsaufruf`* mit diesem Socket durchgeführt werden. Die Funktion `TCP_close` darf auf keinen Fall aufgerufen werden, während noch eine andere TCP-Funktion aktiv ist. Insbesonders wenn Sie den `TCP_receive` und `TCP_send` in verschiedenen Tasks benutzen, müssen Sie darauf achten, dass alle `TCP_`*`funktionsaufrufe`* beendet sind, bevor Sie den `TCP_close` aufrufen. Wenn Sie diese Regeln nicht beachten, können Sie beliebige Fehler bis zum Totalabsturz des Systems erzielen, da Ihre Applikation mit Speicher arbeitet, der u.U. schon von einem anderen Programm genutzt wird!

Nach dem Schließen werden alle vom Netz einlaufenden Pakete für dieses Port zurückgewiesen.

Die Funktion `TCP_close` erwartet folgende Parameter:

Parameter	PEARL-Type	C-Type	Bedeutung
conn	`TCP_STRUCT`	`tcp_socket *`	Socket der Verbindung
errcode	`FIXED(15)`	`Short`	Return-Wert: Fehlercode

10.3.4.1 Beispiel TCP_close

PEARL-Programm:

```
...
DCL conn    TCP_STRUCT;
DCL errcode FIXED(15) ;
...
errcode = TCP_close( conn ) ;
IF errcode GE 0 THEN
   /* fehlerfreie Aktion */
ELSE
```

```
      /* fehlerhafte Aktion */
   FIN;
   ...
```

C-Programm:

```
   ...
   tcp_socket *conn    ;
   short       errcode ;
   ...
   errcode = tcp_close( conn ) ;
   if ( errcode > 0 )
   {
      /* fehlerfreie Aktion */
   }
   else
   {
      /* Fehlerhafte Aktion */
   }
   ...
```

10.3.5 TCP_abort

Die Funktion TCP_abort schließt die Verbindung im Gegensatz zum TCP_close in jedem Fall, d.h. auch wenn die Gegenstation nicht reagiert, es wird nur ein Reset zur Gegenstation geschickt.

Wenn die Funktion fehlerfrei durchgeführt wird, so ist danach der Socket geschlossen, d.h. in der TCP-Socket-Struktur ist der Socketpointer gelöscht.

Bitte beachten Sie, dass auch der Speicher, den die TCP-Socket-Struktur belegt hat, an das Betriebssystem zurück gegeben wurde! Es darf danach kein weiterer TCP_*funktionsaufruf* mit diesem Socket durchgeführt werden. Die Funktion TCP_abort darf auf keinen Fall aufgerufen werden, während noch eine andere TCP-Funktion aktiv ist. Insbesonders wenn Sie den TCP_receive und TCP_send in verschiedenen Tasks benutzen, müssen Sie darauf achten, dass alle TCP_*funktionsaufrufe* beendet sind, bevor Sie den TCP_abort aufrufen. Wenn Sie diese Regeln nicht beachten, können Sie beliebige Fehler bis zum Totalabsturz des Systems erzielen, da Ihre Applikation mit Speicher arbeitet, der u.U. schon von einem anderen Programm genutzt wird!

Nach dem Schließen werden alle vom Netz einlaufenden Pakete für dieses Port zurückgewiesen.

Die Funktion TCP_abort erwartet folgende Parameter:

Parameter	PEARL-Type	C-Type	Bedeutung
conn	TCP_STRUCT	tcp_socket *	Socket der Verbindung
errcode	FIXED(15)	Short	Return-Wert: Fehlercode

10.3.5.1 Beispiel TCP_abort

PEARL-Programm:

```
...
DCL conn    TCP_STRUCT;
DCL errcode FIXED(15) ;
...
errcode = TCP_abort( conn ) ;
IF errcode GE 0 THEN
   /* fehlerfreie Aktion */
ELSE
   /* fehlerhafte Aktion */
FIN;
...
```

C-Programm:

```
...
tcp_socket *conn    ;
short       errcode ;
...
errcode = tcp_abort( conn ) ;
if ( errcode > 0 )
{
   /* fehlerfreie Aktion */
}
else
{
   /* Fehlerhafte Aktion */
}
...
```

10.3.6 TCP_reset und TCP_free_socket

Die Funktion `TCP_reset` schließt die Verbindung im Gegensatz zum `TCP_close` in jedem Fall, d.h. auch wenn die Gegenstation nicht reagiert, es wird nur ein Reset zur Gegenstation geschickt.

Die Kombination der Funktion `TCP_reset und TCP_free_socket` sollte in dem folgenden Fall anstatt eines `TCP_close` eingesetzt werden.

Es wird ein `Socket` von mehreren unabhängigen Tasks genutzt, das heißt Task1 öffnet den `Socket` mit einem `TCP_active_open` (oder `TCP_passive_open`) und übergibt dem `Socket` an eine zweite Task2. Danach nutzt Task1 den `Socket` zum Empfang von Daten mit TCP_receive und die zweite Task2 senden mit Hilfe von TCP_send Daten über den gleichen `Socket`. Im Falls eines Fehlers soll der `Socket` geeignet geschlossen werden, dabei dürfen beide Tasks nicht unabhängig voneinander einen `TCP_close` auslösen, da dann die andere Task auf den schon freigegebenen Speicher des ehemaligen `Socket` zugreifen würde.

Daher sollte im Falle eines Fehlers nur die Funktion `TCP_reset` ausgelöst werden, die dann alle Aufträge dieses `Socket` mit einer Fehlermeldung zurückgibt und erst nach abschliessender Syn-

chronisation der Tasks wird von einer Task ein `TCP_free_socket` ausgelöst um den Speicher des `Socket` freizugeben.

Nach dem `TCP_reset` werden alle vom Netz einlaufenden Pakete für dieses Port zurückgewiesen.

Die Funktion `TCP_reset` erwartet folgende Parameter:

Parameter	PEARL-Type	C-Type	Bedeutung
conn	`TCP_STRUCT`	`tcp_socket *`	Socket der Verbindung
errcode	`FIXED(15)`	`Short`	Return-Wert: Fehlercode

Die Funktion `TCP_free_socket` erwartet folgende Parameter:

Parameter	PEARL-Type	C-Type	Bedeutung
conn	`TCP_STRUCT`	`tcp_socket *`	Socket der Verbindung
Errcode	`FIXED(15)`	`Short`	Return-Wert: Fehlercode

10.3.6.1 Beispiel TCP_reset und TCP_free_socket

PEARL-Programm:

```
...
DCL conn    TCP_STRUCT;
DCL errcode FIXED(15) ;
...
errcode = TCP_reset( conn ) ; /* nur Abbruch Melden */
...
/* warte auf andere Tasks */
REQUEST alles_fertig ; /* Sync mit anderer Task */
errcode = TCP_free_socket( conn ); /* freigeben des Speichers */
...
```

C-Programm:

```
...
tcp_socket *conn    ;
short       errcode ;
...
errcode = tcp_reset( conn ) ; /* Abbruch forcieren */
rt_request_sema( alles_fertig ) ; /* Sync mit anderer Task */
errcode = TCP_free_socket( conn ) ; /* Speicher freigeben */
...
```

10.3.7 TCP_send

Die Funktion `TCP_send` sendet Daten auf das durch `TCP_`*xx*`_open` geöffnete Port. Dabei wird die angegebene Datenlänge (len) komplett an den Netzwerkstack übergeben und übermittelt die Daten. Nur im Falle einer Fehlermeldung (errcode < 0) sind die Daten nicht übertragen. Die Funktion wartet solange bis der Netzwerkstack die Daten komplett übernommen hat.

Die Funktion TCP_send erwartet folgende Parameter:

Parameter	PEARL-Type	C-Type	Bedeutung
conn	TCP_STRUCT	tcp_socket *	Socket der Verbindung
data	TCP_DATA	char *	Pointer auf die Daten
len	FIXED(31)	int	Anzahl Datenbytes
errcode	FIXED(15)	short	Return-Wert: Anzahl gesendete Bytes oder Fehlercode

Besonderheit in PEARL:

Damit beliebige Datentypen als Anwenderdaten übertragen werden können, ist der Datenpointer in einer Struktur versteckt, um beliebige Pointer an die Funktion übergeben zu können.

10.3.7.1 Beispiel TCP_send

PEARL-Programm:

```
TYPE TCP_CHAR STRUCT [ DATA REF CHAR(1)   ] ;
TYPE TCP_FIX  STRUCT [ DATA REF FIXED(15) ] ;
...
DCL BUFF        CHAR(200)  ;
DCL conn        TCP_STRUCT ;
DCL ptr         TCP_CHAR   ;
DCL BUFF_F(100) FIXED(15)  ;
DCL ptrf        TCP_FIX    ;
...
ptr.DATA  = BUFF.CHAR(1) ;
ptrf.DATA = BUFF_F(1)    ;
...
errcode = TCP_send( conn, ptr, 200(31) );
IF errcode GT 0 THEN
   /* Daten erfolgreich uebertragen */
ELSE
  /* Fehlerhafter Status */
FIN;
...
errcode = TCP_send( conn, ptrf, 2*100(31) ) ;
...
```

C-Programm:

```
...
char BUFF[200]   ;
tcp_socket *conn ;
...
errcode = tcp_send( conn, buff, 200L ) ;
```

```
if ( errcode > 0 )
{
   /* Daten erfolgreich uebertragen */
}
else
{
  /* Fehlerhafter Status */
}
...
```

10.3.8 TCP_Nsend

Die Funktion `TCP_Nsend` sendet Daten auf das durch `TCP_xx_open` geöffnete Port. Im Unterschied zu `TCP_send` wird bei der Funktion `TCP_Nsend` auch eine Teilmenge der Daten an den Netzwerkstack übergeben, falls nicht genügend Platz im Sendepuffer ist. Daher kann die Funktion ausser mit einem Fehlercode (errcode < 0) auch mit (errcode < len) antworten, dann sind nur die in `errcode` angegebenen Daten vom Netzwerkstack übernommen worden und der Anwender ist für Wiederholung der restlichen Daten zuständig.

Die Funktion `TCP_Nsend` kommt immer sofort zurück und erzeugt keine internen Wartezeiten, anders als die Funktion `TCP_send`. Der Vorteil der Funktion ist, daß der Anwender den zeitlichen Ablauf seiner Anwendung besser bestimmen kann und geeignete Maßnahmen ergreifen kann, falls die Daten zu langsam übertragen werden.

Die Funktion `TCP_Nsend` erwartet folgende Parameter:

Parameter	PEARL-Type	C-Type	Bedeutung
conn	`TCP_STRUCT`	`tcp_socket *`	Socket der Verbindung
data	`TCP_DATA`	`char *`	Pointer auf die Daten
len	`FIXED(31)`	`int`	Anzahl Datenbytes
errcode	`FIXED(15)`	`short`	Return-Wert: Anzahl gesendete Bytes oder Fehlercode

Besonderheit in PEARL:

Damit beliebige Datentypen als Anwenderdaten übertragen werden können, ist der Datenpointer in einer Struktur versteckt, um beliebige Pointer an die Funktion übergeben zu können.

10.3.8.1 Beispiel TCP_Nsend

PEARL-Programm:

```
TYPE TCP_CHAR STRUCT [ DATA REF CHAR(1)   ] ;
...
DCL BUFF        CHAR(200)  ;
DCL conn        TCP_STRUCT ;
DCL ptr         TCP_CHAR   ;
DCL has_send    FIXED(15)  ;
DCL want_len    FIXED(31)  ;
```

```
DCL pcf     REF CHAR(1)  ;
...
pcf = BUFF.CHAR(1) ; /* auf den Anfang des Puffers */
ptr.DATA  = pcf ;
want_len  = 200(31)      ; /* wir wollen 200 Byte Senden */
...
errcode = TCP_Nsend( conn, ptr, want_len );
IF errcode GE 0 THEN
   has_send = errcode ; /* es wurden (has_send) Daten uebertragen */
   want_len = want_len – has_send ;
   CALL REFADD( pcf, has_send ) ;
   ptr.DATA = pcf ; /* ab hier neu senden */
   WHILE ( want_len GT 0 AND errcode GE 0 )
      ...
      AFTER xxx SEC RESUME ; /* eine Wartezeit */
      errcode = TCP_Nsend( conn, ptr, want_len );
       IF errcode GE 0 THEN
          has_send = errcode ; /*  (has_send) Daten uebertragen */
          want_len = want_len – has_send ;
          CALL REFADD( pcf, has_send ) ;
          ptr.DATA = pcf ; /* ab hier neu senden */
       FIN ;
      ...
   END ;
ELSE
  /* Fehlerhafter Status */
FIN;
...
errcode = TCP_Nsend( conn, ptrf, 2*100(31) ) ;
...
```

C-Programm:

```
...
char BUFF[200]   ;
tcp_socket *conn ;
int want_len     ;
...
want_len = 200 ;
errcode = tcp_Nsend( conn, buff, want_len ) ;
if ((short)errcode >= 0 )
{
   /* ( errcode ) Daten erfolgreich uebertragen */
   want_len -= errcode ;
    do
    {
```

```
          rt_resume_after( 1000 ) ; /* 1 Sec Warten */
          buff += errcode ;
          errcode = tcp_Nsend( conn, buff, want_len ) ;
          if ( (short)errcode >= 0 )
            {
                want_len -= errcode ;
            }
      }
         while( want_len > 0 && errcode >= 0 ) ;
  }
  else
  {
    /* Fehlerhafter Status */
  }
  ...
```

10.3.9 TCP_receive

Die Funktion `TCP_receive` empfängt Daten paketweise von einem durch `TCP_`*xx*`_open` geöffneten Port.

Die Funktion `TCP_receive` erwartet folgende Parameter:

Parameter	PEARL-Type	C-Type	Bedeutung
conn	`TCP_STRUCT`	`tcp_socket *`	Socket der Verbindung
data	`TCP_DATA`	`char *`	Pointer auf die Daten
errcode	`FIXED(15)`	`short`	Return-Wert: Anzahl empfangene Bytes oder Fehlercode

Besonderheit in PEARL:

Damit beliebige Datentypen als Anwenderdaten übertragen werden können, ist der Datenpointer in einer Struktur versteckt, um beliebige Pointer an die Funktion übergeben zu können.

Achtung: Der Datenpuffer muß Platz für min. 1460 Bytes zur Verfügung haben.

10.3.9.1 Beispiel TCP_receive

PEARL-Programm:

```
TYPE TCP_CHAR STRUCT [ DATA REF CHAR(1) ] ;
...
DCL BUFF(6) CHAR(255)  ;
DCL conn    TCP_STRUCT ;
DCL ptr     TCP_CHAR   ;
ptr.DATA = BUFF(1).CHAR(1) ;
errcode = TCP_receive( conn, ptr ) ;
IF errcode GT 0 THEN
```

```
    /*
     * Daten erfolgreich empfangen! In errcode ist die Anzahl
     * der empfangenen Daten verzeichnet
     */
  ELSE
    /* Fehlerhafter Status */
  FIN;
```

C-Programm:

```
  ...
  char        BUFF[1460] ;
  tcp_socket *conn       ;
  ...
  errcode = tcp_receive( conn, BUFF ) ;
  if ( errcode > 0 )
  {
     /* Daten erfolgreich empfangen
      * in errcode ist die Anzahl der
      * empfangenen Daten verzeichnet
      */
  }
  else
  {
    /* Fehlerhafter Status */
  }
```

10.3.10 TCP_Nreceive

Die Funktion `TCP_Nreceive` empfängt Daten von einem durch `TCP_xx_open` geöffneten Port. Im Gegensatz zum `TCP_receive` kann die Länge, die maximal empfangen werden soll, vorgegeben werden.

Die Funktion `TCP_Nreceive` erwartet folgende Parameter:

Parameter	PEARL-Type	C-Type	Bedeutung
conn	`TCP_STRUCT`	`tcp_socket *`	Socket der Verbindung
data	`TCP_DATA`	`char *`	Pointer auf den Datenpuffer
maxlen	`FIXED(31)`	`int`	Puffergröße
errcode	`FIXED(15)`	`short`	Return-Wert: Anzahl empfangene Bytes oder Fehlercode

Besonderheit in PEARL:

Damit beliebige Datentypen als Anwenderdaten übertragen werden können, ist der Datenpointer in einer Struktur versteckt, um beliebige Pointer an die Funktion übergeben zu können.

10.3.10.1 Beispiel TCP_Nreceive

PEARL-Programm:

```
TYPE TCP_CHAR STRUCT [ DATA REF CHAR(1) ] ;
...
DCL BUFF    CHAR(255)  ; /* Puffer für Empfangsdaten */
DCL conn    TCP_STRUCT ;
DCL ptr     TCP_CHAR   ; /* Pointer auf den Empfangspuffer  */
DCL anz     FIXED(15)  ; /* Rueckgabewert, negativ = Fehler */

ptr.DATA = BUFF.CHAR(1) ;
anz = TCP_Nreceive( conn, ptr, 255 ) ;
IF anz GT 0 THEN
   /*
    * Daten erfolgreich empfangen! In anz ist die Anzahl
    * der empfangenen Daten verzeichnet
    */
ELSE
  /* Fehlerhafter Status */
FIN;
```

C-Programm:

```
...
char        BUFF[1460] ;
tcp_socket *conn       ;
short       anz        ;
...
anz = tcp_Nreceive( conn, BUFF, sizeof(BUFF) ) ;
if ( anz > 0 )
{
   /* Daten erfolgreich empfangen
    * in anz ist die Anzahl der
    * empfangenen Daten verzeichnet
    */
}
else
{
  /* Fehlerhafter Status */
}
...
```

10.3.11 TCP_status

Beim Aufruf von `TCP_status` werden die aktuellen Windowgrößen der beiden Kommunikationspartner in der `TCP_STRUCT` eingetragen.

Die Funktion `TCP_status` erwartet folgende Parameter:

Parameter	PEARL-Type	C-Type	Bedeutung
conn	`TCP_STRUCT`	`tcp_socket *`	Socket der Verbindung
errcode	`FIXED(15)`	`short`	Return-Wert: Fehlercode

10.3.11.1 Beispiel TCP_status

PEARL-Programm:

```
...
DCL conn TCP_STRUCT ;
...
errcode = TCP_status( conn ) ;
IF errcode GT 0 THEN
   /* Statusabfrage erfolgreich
    * es sind jetzt in der TCP_STRUCT alle
    * Elemente mit den aktuell gueltigen
    * Verbindungsdaten besetzt
    */
   IF  conn.pSocket.SND_WND GT 1024(31) THEN
      /* falls das Send_window > 1024 Byte */
      ...
   FIN;
ELSE
  /* Statusabfrage nicht erfolgreich */
FIN;
...
```

C-Programm:

```
...
tcp_socket *conn ;
...
errcode = tcp_status( conn ) ;
if ( errcode > 0 )
{
   /* Statusabfrage erfolgreich
    * es sind jetzt in der tcp_scoket alle
    * Elemente mit den aktuell gueltigen
    * Verbindungsdaten besetzt
    */
   if ( conn->snd_wnd > 1024L )
   {
      /* falls das Send_window > 1024 Byte */
      ...
   }
```

```
}
else
{
  /* Statusabfrage nicht erfolgreich */
}
...
```

10.3.12 TCP_set_timeout

`TCP_set_timeout` setzt ein TimeOut für die Funktion `TCP_receive` und `TCP_Nreceive`. Wird innerhalb der gesetzten Zeit kein Paket empfangen, wird `TCP_(N)receive` mit einer entsprechenden Fehlermeldung abgebrochen.

Die TimeOut Zeitbasis sind 512 msec, d.h. ein TimeOut kann nur in 512 msec. Schritten realisiert werden.

Das TimeOut berechnet sich nach folgender Formel:

```
Zeitlimit = ( ( timeout // 512 ) + 1 ) * 512 ;
```

Das Zeitlimit wird also auf die nächste durch 512 teilbare Zahl aufgerundet. Mit dem Wert 0 wird das TimeOut abgeschaltet.

Die Funktion `TCP_set_timeout` erwartet folgende Parameter:

Parameter	PEARL-Type	C-Type	Bedeutung
conn	`TCP_STRUCT`	`tcp_socket *`	Socket der Verbindung
timeout	`FIXED(31)`	`int`	Zeitlimit

10.3.12.1 Beispiel TCP_set_timeout

PEARL-Programm:

```
...
DCL conn TCP_STRUCT ;
...
/* setzen des Zeitlimits auf 1024 msec */
CALL TCP_set_timeout( conn , 1000(31) ) ;
...
/* alle nun folgenden TCP_receive werden nach 1024 msec
 * mit dem Fehlercode ERR_ULP_TIMEOUT beendet, falls kein
 * Paket eingetroffen ist.
 */
...
/* setzen des Zeitlimits auf unbegrenzte Zeit */
CALL TCP_set_timeout( conn , 0(31) ) ;
...
```

C-Programm:

```
...
```

```
tcp_scoket *conn ;
...
/* setzen des Zeitlimits auf 1024 msec */
tcp_set_timeout( conn , 1000L ) ;
...
/* alle nun folgenden TCP_receive werden nach 1024 msec
 * mit dem Fehlercode ERR_ULP_TIMEOUT beendet, falls kein
 * Paket eingetroffen ist.
 */
...
/* setzen des Zeitlimits auf unbegrenzte Zeit */
tcp_set_timeout( conn , 0L ) ;
...
```

10.3.13 TCP_cleanup

Mit der Funktion `TCP_cleanup` wird der TCP-Protokollsoftware mitgeteilt, **alle** aktiven Verbindungen zu schließen. Die Funktion `TCP_cleanup` hat keine Parameter.

10.3.13.1 Beispiel TCP_cleanup

PEARL-Programm:

```
...
/* TCP-Protokollstack zurücksetzen */
errcode = TCP_cleanup ;
...
```

C-Programm:

```
...
/* TCP-Protokollstack zurücksetzen */
errcode = tcp_cleanup() ;
...
```

10.3.14 Fehlercodes des TCP

Der Rückgabeparameter der einzelnen TCP-Funktionen sollte nach jedem Funktionsaufruf abgefragt werden. Eine erfolgreiche Operation gibt den Fehlercode >= 0 zurück.

Ein Fehlercode <0 ergibt einen Fehlerstatus der folgendermaßen decodiert werden kann:

PEARL-Programm:

```
...
DCL ret_val FIXED(15) ;
DCL errcode FIXED(15) ;
...
ret_val = TCP_funktion(...);
IF ret_val LT 0 THEN
```

```
   /* Fehlerhafte Operation */
   errcode = TOFIXED( TOBIT(ret_val) AND '7FFF'B4 ) ;
   /* Bedeutung der errcode siehe Tabelle TCP-Fehlermeldung */
ELSE
  /* Fehlerfreie Operation */
FIN;
...
```

C-Programm:

```
...
short ret_val ;
short errcode ;
...
ret_val = tcp_funktion(...);
if ( ret_val < 0 )
{
   /* Fehler, Bedeutung des errcode siehe Tabelle TCP-Fehlermeldung */
}
else
{
  /* Fehlerfreie Operation */
}
...
```

10.3.15 TCP-Fehlermeldungen und Statuskodes

Der Wert `Err` wird zurückgegeben. Wird das oberste Bit gelöscht, erhält man die `Nr`.

Fehler	Nr	Err	Bedeutung
ERR_PCB_PORT_0	1	0x8001	SourcePort = 0, nicht erlaubt
ERR_PCB_OVERFLOW	2	0x8002	Alle Ports geöffnet
ERR_PCB_WRONG_FILENAME	3	0x8003	Port nicht geöffnet
ERR_PCB_WRONG_PORT	4	0x8004	falsche Portnummer beim Close
ERR_PCB_FILENAME_IN_USE	5	0x8005	Port schon geöffnet
ERR_PCB_PORT_CLOSED	6	0x8006	Port ist geschlossen
ERR_PROTO_NIMPL	16	0x8010	Protokoll nicht implementiert
ERR_NO_ARP_RESP	48	0x8030	keine ARP-Antwort
ERR_OWN_ETH_ADR_UNKNOWN	53	0x8035	Hardwaretreiber nicht gestartet?
ERR_ULP_TIMEOUT	64	0x8040	Read Timeout
ERR_TCP_CON_EXISTS	128	0x8080	Verbindung schon existend
ERR_TCP_CON_NOT_EXISTS	129	0x8081	Verbindung nicht existend
ERR_TCP_ILLEGAL_REQUEST	130	0x8082	Fehlerhafte Anfrage
ERR_TCP_CON_CLOSING	131	0x8083	Verbindung wird geschlossen

ERR_TCP_INSUFF_RESOURCE	132	0x8084	Sendedaten nicht übermittelt
ERR_TCP_REQERR	133	0x8085	falsche Requestnummer
ERR_TCP_PACKET_LONG	134	0x8086	Paket > 1432 Bytes
ERR_TCP_STATE_WRONG	135	0x8087	Status nicht erlaubt

10.4 User Datagramm Protocol UDP

10.4.1 Aufgaben des UDP

Über UDP wird eine Punkt zu Punkt Verbindung zwischen zwei Kommunikationspartnern aufgebaut. Sie ist durch vier Parameter eindeutig festgelegt:

```
Quell-Adresse, Ziel-Adresse, Quell-Port, Ziel-Port
```

Die Adresse legt den angesprochenen Rechner fest, das Port bestimmt die Anwendung auf dem Rechner. Die Adresse ist eine eindeutige IP-Adresse, die Portnummer sollte für eigene Anwendungen im Bereich zwischen 1024 und 32767 liegen. Unter 1024 liegen Ports, die für bestimmte Anwendungen reserviert sind. Die Anzahl der maximal gleichzeitig geöffneten Ports ist nur durch den zur Verfügung stehenden Speicher begrenzt.

Die Anwenderdaten werden in UDP-Paketen übertragen. Die Länge pro Paket darf zwischen 1 und 1470 Byte liegen.

UDP stellt keine gesicherte Verbindung zur Verfügung, d.h. es können Pakete verloren gehen oder in der Reihenfolge vertauscht sein! Empfangene Pakete enthalten aber in jedem Fall korrekte Daten. Sie als Programmierer der Anwendung müssen dafür sorgen, dass Ihre Anwendung damit zurecht kommt. Sonst können Sie auf TCP ausweichen, wo Ihnen eine gesicherte Verbindung zur Verfügung steht.

10.4.1.1 Aufbau eines UDP-Paketes

Für die Datenübertragung werden UDP-Pakete genutzt, diese haben einen festgelegten Aufbau. Sie als Anwender brauchen sich um diesen Aufbau nicht kümmern, der UDP-Stack erzeugt entsprechende Pakete.

Ein Paket besteht aus dem Header- und dem Data-Teil. Im Header sind die für die Netzwerkverwaltung benötigten Daten abgelegt :

```
Hardware-Ziel-Adresse   /* vom UDP-Stack erzeugt */
Hardware-Quell-Adresse  /*            "          */
Pakettype               /*            "          */
IP-Header               /*            "          */
UDP-Header              /* über UDP-Funktion     */
Userdaten               /* Anwenderdaten         */
```

Im Data-Teil eines Paketes sind die vom Anwender zu übermittelnden Daten abgelegt. Pro Paket können minimal 1 Byte und maximal 1470 Byte übertragen werden. Pakete mit anderen Längen werden nicht akzeptiert und mit einer Fehlermeldung vom UDP-Stack abgewiesen.

10.4.2 Die UDP-Programmierschnittstelle

Die folgende Beschreibung ermöglicht es Ihnen eigene UDP-Verbindungen zu programmieren. Als Programmiersprache werden PEARL und CREST-C unterstützt.

10.4.2.1 UDP-Library

Für die Sprache PEARL steht keine eigene Library zur Verfügung. Die erforderlichen Funktionen sind im PEARL-Beispielprogramm enthalten (Datei p_udp.p). Für CREST-C sind alle Programme mit der NETLIB.LIB zu linken.

10.4.2.2 UDP-Paket

Ein UDP-Paket besteht aus einem Header- und einem Daten-Teil. Der Aufbau des Headers ist festgelegt, den Datenteil können Sie nach Ihren Wünschen anlegen. Nachfolgend eine mögliche Beschreibung eines Paketes:

PEARL	CREST-C	Kommentar
`TYPE UDP_PACKET STRUCT`	`struct udp_packet`	
`[`	`{`	
`  HEAD  UDP_HEAD`	`  udp_head header ;`	`/* UDP-Header */`
`  PUFFER  DATA`	`  char buffer[1470] ;`	`/* Daten */`
`] ;`	`};`	

Die DATA-Struktur in PEARL könnte z.B. folgenden Aufbau haben:

```
TYPE DATA STRUCT
[
  DATA1  CHAR(250),     /* Daten 1470 Byte */
  DATA2  CHAR(250),
  DATA3  CHAR(250),
  DATA4  CHAR(250),
  DATA5  CHAR(250),
  DATA6  CHAR(220)
];
```

Typvereinbarung für den UDP-HEADER :

PEARL	CREST-C	Kommentar
`TYPE UDP_HEAD STRUCT`	`struct udp_head`	
`[`	`{`	
`  REQUEST  FIXED(15)`	`  short request ;`	`/*gewuenschte Funktion*/`
`  SOURCE_PORT  FIXED(15)`	`  short source_port ;`	`/*Quell-Port */`
`  DEST_IP_ADDR BIT(32)`	`  int dest_addr ;`	`/*Ziel IP-Adresse */`
`  DEST_PORT  FIXED(15)`	`  short dest_port ;`	`/*Ziel-Port */`
`  DATA_LENGTH  FIXED(31)`	`  int datalength ;`	`/*Datenlaenge */`
`] ;`	`};`	

10.4.2.2.1 UDP Headerparameter bei PEARL

Im UDP-Header sind die Übergabeparameter zwischen dem PEARL-Programm und dem UDP untergebracht. Sie dienen zur Steuerung der UDP-Funktionalität.

HEAD.REQUEST	0 = Öffnen eines Ports 1 = Schreiben auf einem Port 2 = Lesen von einem Port 3 = Schliessen eines Ports
HEAD.SOURCE_PORT	Port der Task auf dem Rechner, die den UDPOPEN absetzt
HEAD.DEST_IP_ADDR	IP-Adresse des Empfangsrechners
HEAD.DEST_PORT	Port der Task auf dem Empfangsrechner
HEAD.DATA_LENGTH	Anzahl der Anwenderdaten in diesem Paket

UDP-Socket in C:

Das CREST-C-Programm kommuniziert mit den UDP-Funktionen über einen Socket. In diesem Socket sind alle verbindungsrelevanten Daten vermerkt. Aufbau:

```
struct udp_socket
{
  u_long  dest_addr   ;
  u_short source_port ;
  u_short dest_port   ;
  int     time_out    ;
  char    f_name[32]  ;
}
typedef struct udp_socket udp_socket ;
```

Zum Aufbau einer eindeutigen Verbindung müssen folgende Bedingungen eingehalten werden:

Quelle (Sender)		Ziel (Receiver)
`Source_Port`	=	`Destination_Port`
`Destination_Port`	=	`Source_Port`
`Destination_IP`	=	`eigene IP`

Soll ein Server realisiert werden, der einen Verbindungswunsch von einem beliebigen Rechner und /oder Port entgegen nehmen kann, so dürfen beim Öffnen `Destination_IP` und/oder `Destination_Port` auf 0 gesetzt werden. Soll ein Broadcast verschickt werden, so darf `Destination_IP` auf `0xFFFFFFFF` (=255.255.255.255) gesetzt werden.

10.5 UDP Funktionen

Für das UDP stehen die folgenden Funktionen zur Verfügung:

Funktion	Bedeutung
UDP_open	Aufbau einer UDP-Verbindung
UDP_close	Schliessen einer Verbindung
UDP_send	Senden von Daten über die Verbindung
UDP_receive	Empfangen von Daten
UDP_set_timeout	Setzen des Empfangstimeout
UDP_cleanup	Löschen von Receive-Paketen nach close (nur CREST-C)

Die Verbindung des durch UDP_open geöffneten Ports zum Programm erfolgt über die Struktur Socket. In dieser Struktur werden interne Zustände des Protokollstacks vermerkt, deshalb darf der Socket zwischen einem UDP_open und einem UDP_close nicht verändert werden! Weiterhin sind eindeutige Filenamen beim UDP_open zu verwenden!

Die PEARL-Statements OPEN, CLOSE wirken auf das UDP in folgender Weise:

```
OPEN dat BY IDF(...),ANY        löscht alle alten Pakete
CLOSE dat BY CAN                löscht alle alten Pakete
OPEN dat BY IDF(...)            ändert nur den Namen
```

10.5.1 UDPOPEN

Die Funktion UDP_open erwartet folgende Parameter:

PEARL:

Parameter	PEARL-Type	Bedeutung
Dation	DATION	Datenstation des PEARL-Programms
UDP_struct	UDP_PACKET	UDP Datenstruktur
S_Port	FIXED(15)	Source Port auf dem eigenen Rechner
D_IP	BIT(32)	IP-Adresse des Zielrechners
D_PORT	FIXED(15)	Portnummer auf dem Zielrechner

Die Funktion UDPOPEN stellt die Verbindung zwischen einer PEARL-Dation und einem UDP-Port her. Somit können vom PEARL-Programm in gewohnter Weise Daten mit Hilfe der Funktionen READ/WRITE übertragen werden.

CREST-C:

Parameter	C-Type	Bedeutung
UDP_struct	udp_packet *	UDP Datenstruktur
D_IP	u_long	IP-Adresse des Zielrechners
S_Port	u_short	Source Port auf dem eigenen Rechner
D_Port	u_short	Portnummer auf dem Zielrechner
F_Name	char *	Filename der Verbindung
socket	udp_socket *	Rückgabewert: <> 0 wenn gültiger Socket

Für das jeweilige Anwender-Programm werden die einzelnen geöffneten UDP-Ports mit Hilfe des Filenamen unterschieden. Daher muß für jeden Zugriff auf ein Port der **eindeutige** Filename genutzt werden. Diese Zuordnung wird erst beim Schließen der Verbindung getrennt.

Durch die Angabe einer eindeutigen IP-Adresse ist der zugehörige Kommunikationspartner festgelegt, einzige Ausnahme ist die Nutzung der Broadcast-IP-Adresse (255.255.255.255) bei der die Pakete an alle im Netz befindlichen Rechner (außer dem eigenen) gesendet werden. Die Broadcast-IP-Adresse darf nur beim Senden genutzt werden.

Für den Empfang von Paketen kann das UDP unspezifiziert geöffnet werden, dazu müssen die Destination-IP-Adresse und das Destinationport auf NULL gesetzt werden. Nun werden alle Pakete, die der angegebenen Sourceportnummer entsprechen, akzeptiert und können eingelesen werden.

10.5.1.1 Beispiele UDPOPEN

Öffnen eines unspezifizierten Ports:

PEARL-Programm:

```
...
CALL UDPOPEN( dat, udp, 1024, '00000000'B4, 0 );
IF ST( dat ) EQ 0 THEN
   /* Öffnen o.k., Daten verschicken */
   ...
ELSE
   PUT 'Open Error in UDP' TO TYX BY A,SKIP;
FIN;
...
```

C-Programm:

```
...
udp_socket *conn    ;
udp_packet  my_pack ;
...
conn = udp_open( &my_pack, 0x00000000L, 0, 1024, "port1400" ) ;
if ( conn != NULL )
{
   /* socket konnte erzeugt werden */
   ...
   /* Datentransfer */
}
else
{
   /* es konnte kein Socket erzeugt werden */
}
```

Lesen:

Alle Pakete, die vom Netz einlaufen und als Destinationport 1024 eingetragen haben, werden vom UDP akzeptiert und können von der Task eingelesen werden. Aus dem `UDP-HEADER` kann der Absender ermittelt werden.

Schreiben:

Ist auf eine so geöffnete Verbindung nicht erlaubt.

Öffnen eines Ports:

PEARL-Programm:

```
...
CALL UDPOPEN( dat, udp, 1200, 'C1010101'B4, 1300 );
IF ST( dat ) EQ 0 THEN
   /* Öffnen o.k., Daten verschicken */
   ...
ELSE
   PUT 'Open Error in UDP' TO TYX BY A,SKIP;
FIN;
...
```

C-Programm:

```
...
udp_socket *conn     ;
udp_packet  my_pack  ;
...
conn = udp_open( &my_pack, 0xC1010101L, 1200, 1300, "port1400" ) ;
if ( conn != NULL )
{
   /* socket konnte erzeugt werden */
   ...
   /* Datentransfer */
}
else
{
   /* es konnte kein Socket erzeugt werden */
}
```

Lesen:

Vom UDP werden nur Pakete des Absenders mit der IP-Adresse `'C1010101'B4` (entspricht `193.1.1.1`) und `Sourceport`=1300 sowie eingetragenem `Destination-port`=1200 akzeptiert. Alle Pakete, die diese Restriktionen nicht erfüllen, werden nicht an die PEARL-Ebene durchgereicht.

Schreiben:

Alle auf die Dation `dat` ausgegebenen Pakete werden vom UDP an die Station `'C1010101'B4` (entspricht `193.1.1.1`) verschickt, dabei ist als `Sourceport`=1200 und `Destinationport`=1300 eingetragen.

Öffnen eines Ports zum Broadcasts verschicken:

PEARL-Programm:

```
...
CALL UDPOPEN( dat, udp, 1025, 'FFFFFFFF'B4, 1024 );
IF ST( dat ) EQ 0 THEN
   /* Öffnen o.k., Daten verschicken */
   ...
ELSE
   PUT 'Open Error in UDP' TO TYX BY A,SKIP;
FIN;
...
```

C-Programm:

```
...
udp_socket *conn     ;
udp_packet  my_pack  ;
...
conn = udp_open( &my_pack, 0xFFFFFFFFL, 1200, 1300, "port1400" ) ;
if ( conn != NULL )
{
   /* socket konnte erzeugt werden */
   ...
   /* Datentransfer */
}
else
{
   /* es konnte kein Socket erzeugt werden */
}
```

Lesen:

Ist von einer so geöffneten Verbindung nicht erlaubt.

Schreiben:

Alle auf die Dation `dat` ausgegebenen Pakete werden vom UDP an alle Stationen des Netzes verschickt, dabei ist `Sourceport`=1025 und `Destinationport`=1024 eingetragen.

10.5.2 UDPCLOSE

Die Funktion UDPCLOSE löscht die Verbindung zwischen Ihrem Programm und dem UDP-Port. Dem UDP wird hiermit der Auftrag erteilt keine weiteren vom Netz einlaufenden Pakete für dieses Port zu akzeptieren.

Die Funktion UDPCLOSE erwartet folgende Parameter:

PEARL:

Parameter	PEARL-Type	Bedeutung
Dation	`DATION`	Datenstation des PEARL-Programms
UDP_struct	`UDP_PACKET`	UDP Datenstruktur

CREST-C:

Parameter	C-Type	Bedeutung
UDP_struct	`udp_packet *`	UDP Datenstruktur
Socket	`udp_socket *`	UDP Socket
errcode	`short`	Rückgabewert: Fehlercode

10.5.2.1 Beispiele UDPCLOSE

Schließen eines Portes:

PEARL-Programm:

```
CALL UDPCLOSE( dat, udp );
```

C-Programm:

```
...
udp_socket *conn     ;
udp_packet  my_pack  ;
short       errcode  ;
...
errocde = udp_close( conn , &my_pack );
if ( errcode > 0 )
{
   /* Fehlerfreie Aktion, der Socket ist geschlossen */
   ...
}
else
{
   /* Fehlerhafte Aktion */
}
```

10.5.3 UDPSEND

Die Funktion UDPSEND sendet Daten auf das durch UDPOPEN geöffnete Port. Die logische Verbindung zwischen dem Port und der angegebenen Dation nimmt das UDP über den Dation-Filenamen vor.

Die Funktion UDPCLOSE erwartet folgende Parameter:

PEARL:

Parameter	PEARL-Type	Bedeutung
Dation	`DATION`	Datenstation des PEARL-Programms
UDP_struct	`UDP_PACKET`	UDP Datenstruktur

Vor dem Aufruf von `UDPSEND` müssen folgende Parameter in der Struktur `UDP_PACKET` gesetzt werden:

`UDP.HEAD.DATA_LENGTH` = Anzahl der zu übermittelnden Daten

`UDP.PUFFER.DATA1.`*x* = Datenpuffer sollte mit Anwenderdaten gefüllt sein

CREST-C:

Parameter	C-Type	Bedeutung
UDP_struct	`udp_packet *`	UDP Datenstruktur
Socket	`udp_socket *`	UDP Socket
errcode	`short`	Rückgabewert: Fehlercode

Vor dem Aufruf von `udp_send` müssen die zu übertragenden Daten in das Datenfeld der Struktur `udp_packet` kopiert werden.

10.5.3.1 Beispiel UDPSEND

Senden von Daten auf einem Port:

PEARL-Programm:

```
UDP.HEAD.DATA_LENGTH = 120 ; /* 120 Byte Anwenderdaten */
FOR I TO 120 REPEAT;
   UDP.PUFFER.DATA1.CHAR(I) = 'X';
END;

CALL UDPSEND(dat,udp);
IF ST( dat ) EQ 0 THEN
   /* Daten erfolgreich verschickt */
   ...
ELSE
   /* Daten nicht verschickt -> Fehler */
...
FIN;
...
```

C-Programm:

```
...
udp_socket *conn    ;
udp_packet  my_pack ;
short       errcode ;
int         len     ;
...
len = 100 ;
strcpy( my_pack.buffer , "HALLO DU DA" );
...
errcode = tcp_send( conn , &my_pack , len );
if ( errcode > 0 )
{
   /* Daten versendet */
}
else
{
   /* Fehler aufgetreten */
}
```

10.5.4 UDPREAD

Die Funktion UDPREAD empfängt Daten auf dem durch UDPOPEN geöffneten Port. Die logische Verbindung zwischen dem Port und der angegebenen Dation nimmt das UDP über den Dation-Filenamen vor.

Die Funktion UDPREAD erwartet folgende Parameter:

PEARL:

Parameter	PEARL-Type	Bedeutung
Dation	`DATION`	Datenstation des PEARL-Programms
UDP_struct	`UDP_PACKET`	UDP Datenstruktur

Nachdem dem Einlaufen eines Paketes können die Daten aus der Struktur `UDP_PACKET` gelesen werden:

`UDP.HEAD.DATA_LENGTH` = Anzahl der empfangenen Daten
`UDP.PUFFER.DATA1.`*`x`* = Datenpuffer ist mit Anwenderdaten gefüllt

Bitte beachten Sie: Wird beim `ST(`*`dat`*`)` eine 1 zurückgeliefert, so ist dies kein Fehler, sondern es wurden weniger Daten geliefert, als der Puffer groß ist.

CREST-C:

Parameter	C-Type	Bedeutung
UDP_struct	`udp_packet *`	UDP Datenstruktur
Socket	`udp_socket *`	UDP Socket
errcode	`short`	Rückgabewert: Fehlercode

Nach dem Einlaufen eines Paketes können die Daten aus der Struktur `udp_packet` kopiert werden.

10.5.4.1 Beispiel UDPREAD

Empfangen von Daten auf einem Port:

PEARL-Programm:

```
CALL UDPREAD(dat,udp);
IF ST( dat ) EQ 0 OR ST( dat ) EQ 1 THEN
   /* Daten erfolgreich empfangen */
   FOR I TO UDP.HEAD.DATA_LENGTH REPEAT;
      /* Anzeigen der eingegangenen Daten */
      PUT UDP.PUFFER.DATA1.CHAR(I) TO OUTPUT BY A(1);
   END;
   ...
ELSE
   /* Daten nicht empfangen -> Fehler */
...
FIN;
...
```

C-Programm:

```
...
udp_socket  *conn    ;
udp_packet   my_pack ;
short        errcode ;
short        count   ;
...
errcode = udp_receive( conn , &my_pack );
if ( errcode > 0 )
{
   /* datenpaket  eingelaufen */
   count = my_pack.header.datalength ;/* Anzahl der Daten */
   ...
}
else
{
   /* Fehlerhafte Aktion */
}
```

10.5.5 UDP_SET_TIMEOUT

Mit der Funktion UDP_SET_TIMEOUT kann für diese Verbindung die Wartezeit auf einlaufende Pakete für ein UDPREAD limitiert werden.

Das Zeitraster beträgt ein Vielfaches von 512 msec.

Das wirksame Timeout berechnet sich nach folgender Formel:

```
Zeitlimit = ( ( timeout // 512 ) + 1 ) * 512 ;
```

Die Funktion UDP_SET_TIMEOUT erwartet folgende Parameter:

PEARL:

Parameter	PEARL-Type	Bedeutung
Dation	`DATION`	Datenstation des PEARL-Programms
timeout	`FIXED(31)`	Time-Out Zeitdauer

CREST-C:

Parameter	C-Type	Bedeutung
Socket	`udp_socket *`	UDP Socket
timeout	`int`	Time-Out Zeitdauer

10.5.5.1 Beispiel UDP_SET_TIMEOUT

Setzen des Read Zeitlimits:

PEARL-Programm:

```
...
DCL conn UDP_STRUCT ;
...
/* setzen des Zeitlimits auf 1024 msec */
CALL UDP_SET_TIMEOUT( conn , 1024(31) ) ;
...
/*
 * alle nun folgenden UDPREAD werden nach der Timeoutzeit
 * mit dem Fehlercode ERR_ULP_TIMEOUT beendet, falls kein
 * neues Paket eingetroffen ist
 */
...
/* setzen des Zeitlimits auf unbegrenzt Warten */
CALL UDP_SET_TIMEOUT( conn , 0(31) ) ;
...
```

C-Programm:

```
...
udp_socket *conn ;
...
/* setzen des Zeitlimits auf 1024 msec */
udp_set_timeout( conn , 1000L ) ;
...
/*
 * alle nun folgenden udp_receive werden nach der Timeoutzeit
 * mit dem Fehlercode ERR_ULP_TIMEOUT beendet, falls kein
```

```
 * neues Paket eingetroffen ist
 */
...
/* setzen des Zeitlimits auf unbegrenzt Warten */
udp_set_timeout( conn , 0L ) ;
...
```

10.5.6 UDP_CLEANUP

Die Funktion `UDP_cleanup` erwartet keine Parameter! Sie wird nur von CREST-C aus unterstützt.

Diese Funktion kann aufgerufen werden, um UDP-Pakete, die nach einem `UDP_close` eingelaufen sind, zu löschen.

10.5.7 Fehlercodes des UDP

Bei den einzelnen UDP-Funktionen wird im Falle eines Fehlers bei der zugehörigen Dation der ST-Parameter gesetzt (PEARL), bzw. der entsprechende Fehlercode zurück gegeben (CREST-C). Daher sollte nach jedem Funktionsaufruf der Status der Operation abgefragt werden. Eine erfolgreiche Operation gibt den Fehlercode 0 zurück. Für PEARL gilt, dass zusätzlich auf die Konsole eine entsprechende Fehlermeldung ausgegeben wird, diese kann mit dem NE-Flag bei der Dation abgeschaltet werden.

Fehler	ST	Err	Bedeutung
ERR_PCB_PORT_0	1	0x8001	SourcePort = 0, nicht erlaubt
ERR_PCB_OVERFLOW	2	0x8002	Alle Ports geöffnet
ERR_PCB_WRONG_FILENAME	3	0x8003	Port nicht geöffnet
ERR_PCB_WRONG_PORT	4	0x8004	falsche Portnummer beim Close
ERR_PCB_FILENAME_IN_USE	5	0x8005	Port schon geöffnet
ERR_PCB_PORT_CLOSED	6	0x8006	Port ist geschlossen
ERR_PROTO_NIMPL	16	0x8010	Protokoll nicht implementiert
ERR_UDP_REQERR	32	0x8020	falsche Request Nummer
ERR_UDP_PACKET_LONG	33	0x8021	Paket zu lang (>1470 Bytes)
ERR_NO_ARP_RESP	48	0x8030	keine ARP-Antwort
ERR_OWN_ETH_ADR_UNKNOWN	53	0x8035	Hardwaretreiber nicht gestartet?
ERR_NET_UNREACHABLE	54	0x8036	Es gibt keine Route zu dem Netz
ERR_ULP_TIMEOUT	64	0x8040	Read Timeout

11 Beispieldateien

Mit der Netzwerksoftware werden folgende Beispieldateien mitgeliefert:

`h\error.h`	Fehlercodes
`h\proto.h`	Prototypen
`h\socket.h`	Typdefinition
`tcp\c\tcp_demo.c`	Sende und Empfangstasks für TCP-Kommunikation
`tcp\p\daytime.p`	Server für Datum und Uhrzeit
`tcp\p\p_tcp.p`	Sende und Empfangstasks für TCP-Kommunikation
`udp\c\rmess.c`	Empfangen von UDP-Daten
`udp\c\tmess.c`	Senden von UDP-Daten
`udp\p\beispiel.p`	Versenden und empfangen von UDP-Daten
`udp\p\p_udp.p`	Erweiterte Version mit Shellmodulen (FileXfer ...)