Protocol Interface Manual

TCP/IP
UDP/IP

Hilscher Gesellschaft für Systemautomation mbH
Rheinstraße 15
D-65795 Hattersheim
Germany

Tel. +49 (6190) 9907 - 0
Fax. +49 (6190) 9907 - 50

Sales: +49 (6190) 9907 - 0
Hotline and Support: +49 (6190) 9907 - 99

Sales email: sales@hilscher.com
Hotline and Support email: hotline@hilscher.com

Web: http://www.hilscher.com

Index	Date	Firmware Version	Chapter	Revision
1	1.12.00	1.000	All	Initial release
2	27.1.00	1.001	3.2	Corrected length of TCP_UDP_CMD_CLOSE answer message Added TCP_UDP_CMD_CLOSE_ALL command
3	29.6.01	1.100	2.1 3 4 5.2 5.2.6 5.2.7	Added handshake mode, watchdog time, and Ethernet address Added section ‘Start-up of the Device’ Added section ‘Sockets-based Programming Model’ Changed maximum timeout values Corrected msg.ln value of TCP_CMD_SEND command Corrected data part of UDP_CMD_SEND answer
4	8.8.01	1.200	5.1.7 5.2.2, 5.2.3 5.2.5 5.2.6 5.2.7 5.2.8, 5.2.9 5.2.10 5.2.11 6.1.2 6.2.2 7	Added response time and error code 118 in answer message Added description of special error codes Added error code 119 in answer message Added desciption of special error codes Added error codes 115, 131, and 132 to answer message Added desciption of special error codes Added description of sequenced messgage transfer Corrected msg.ln value of answer message Added keep-alive option Restricted Inactivity Timeout to TCP sockets Added description of keep-alive or inactivity timeout failure Added description of msg.e field Changed command code to 57 Added error code 118 Added error code 132 Added technial data
5	6.2.02	1.210	2.1.3, 5.1.2, 5.1.3 5.1.6, 5.2.8, 5.2.9 6.2.1 6.2.2 7	Added 100 MBit/s flag and auto-negotiation flag Added IP multicast options Replaced error code 55 by error code 218 Added error code 133 Added IP multicast capabilities Added supported RFCs

Although this program has been developed with great care and intensively tested, Hilscher Gesellschaft für Systemautomation mbH cannot guarantee the suitability of this program for any purpose not confirmed by us in writing.

Guarantee claims shall be limited to the right to require rectification. Liability for any damages which may have arisen from the use of this program or its documentation shall be limited to cases of intent.

We reserve the right to modify our products and their specifications at any time in as far as this contributes to technical progress. The version of the manual supplied with the program applies.

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

1 Introduction

This manual describes the user interface of TCP/IP and UDP/IP protocol for the communication interface and the communication module. The aim of this manual is, to support the integration of these devices into own applications based on driver functions or direct access to the dual-port memory.

The general mechanism of data transfer, for example how to send and receive a message or how to perform a warmstart, is independent from the protocol. These procedures are common to all devices, and are described in the Toolkit Manual ‘General Definitions’.

1.1 Definitions

All variables, parameters, and data used in this manual have basically the LSB/MSB (“Intel”) data representation. This corresponds to the convention of the Microsoft C Compiler.

Values followed by ‘h’ are in hexadecimal notation such as 1Eh = 30 decimal. Values without any following letter are in decimal notation.

All IP addresses in this document have host byte order.

1.2 Protocol Signification

To manage the TCP/IP and UDP/IP protocol, two tasks are involved in the system. Therefore following entries for the protocol signification in the variables `TaskiName` are done:

```
Task6Name: ‘TCP_UDP’
Task7Name: ‘IP’
```

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

2 Protocol Parameters

The IP task needs some basic configuration data to be able to start. Normally, these configuration data will be read from a database residing in the device's flash memory. The configuration tool SyCon can be used to create such a database and store it into the flash memory.

It is, however, also possible to provide the configuration data dynamically using an application program. Configuration data sent to the device this way will have precedence over the data read from flash, but will not be stored in flash memory. Hence, the dynamic configuration procedure must be executed again after each reset or power cycle of the device.

2.1 Providing Parameters in Configuration Area of the Dual-Port Memory

2.1.1 Using Device Driver Functions

To provide the configuration data use the device driver function `DevPut-TaskParameter`.

```
short DevPutTaskParameter(unsigned short  usDevNumber,
                          unsigned short  usNumber,
                          unsigned short  usSize,
                          void           *pvData);
```

For parameter `usNumber` use the value 2. Set parameter `usSize` to 24, in order to determine the length of the structure. Point the parameter `pvData` to a data area containing your configuration data. Please see the structure's definition:

```
typedef struct IP_CONFIG_DPMtag {

  unsigned char   bPlcMode;
  unsigned short  usWatchdogTime;
  unsigned char   bReserved3;

  unsigned short  usFlags;
  unsigned long   ulIpAddr;
  unsigned long   ulNetMask;
  unsigned long   ulGateway;
  unsigned char   abEthernetAddr[6];

} IP_CONFIG_DPM;
```

To bring the new configuration data into effect, a warm-start of the device must be performed using the `DevReset` function:

```
short DevReset(unsigned short usDevNumber,
               unsigned short usMode,
               unsigned long  ulTimeout);
```

Set the `usMode` parameter to `WARMSTART` and give a timeout of 5000 ms in the `ulTimeout` parameter.

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

2.1.2 Direct Write Access to the Dual-Port Memory

The configuration data can also directly be written to the dual-port memory's configuration area. Please see the table below for the location of the appropriate variables.

Variable	Data Type	Offset Address in Dual-Port Memory
bPlcMode	USIGN8	01EC0h
usWatchdogTime	USIGN16	01EC1h
bReserved3	USIGN8	01EC3h
usFlags	USIGN16	01EC4h
ulIpAddr	USIGN32	01EC6h
ulNetMask	USIGN32	01ECAh
ulGateway	USIGN32	01ECEh
abEthernetAddr	USIGN8[6]	01ED2h

Once all configuration data are filled in, a warm-start command must be issued to the device to bring the new parameters into effect. Please, refer to the section 'Initialization of the Device' in the 'Toolkit Manual - General Definitions' for a step-by-step description of the warm-start procedure.

2.1.3 Description of the Protocol Parameters

- **Parameter `bPlcMode`**

The `bPlcMode` variable defines the handshake mode to be used during process data exchange with the application program. Its value will not be used by the TCP_UDP task or IP task themselves, but will be passed to the task handling the higher layer protocol. The sections 'IO Communication with a Process Image' and 'Handshake Modes for the Process Data' of the 'Toolkit Manual - General Definitions' provide a detailed description of the handshake mechanism.

- **Parameter `usWatchdogTime`**

The `usWatchdogTime` parameter configures the watchdog timer, which can be used to supervise the application program. Please, refer to the section 'Supervision' of the 'Toolkit Manual - General Definitions' for a detailed description of the watchdog mechanism.

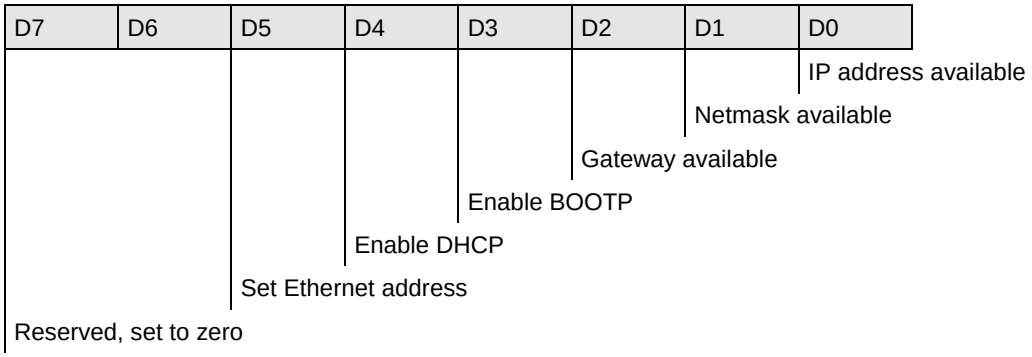
- **Parameter `bReserved3`**

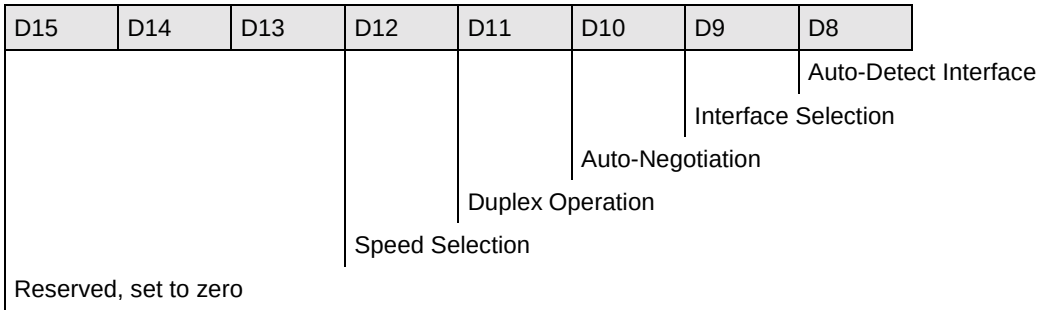
The `bReserved3` parameter marks a reserved area, which will be used for future enhancements. Its content should not be changed.

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

- **Parameter `usFlags`**

The `usFlags` variable holds bit-oriented flags according to the following table:





IP address available (`IP_CFG_FLAG_IP_ADDR`)
If set, the content of the `ulIpAddr` variable will be evaluated.

Netmask available (`IP_CFG_FLAG_NET_MASK`)
If set, the content of the `ulNetMask` variable will be evaluated. If the flag is not set the the device will assume to be an isolated host which is not connected to any network. The `ulGateway` variable will be ignored in this case.

Gateway available (`IP_CFG_FLAG_GATEWAY`)
If set, the content of the `ulGateway` variable will be evaluated. If the flag is not set the the device will assume that there exists no gateway.

Enable BOOTP (`IP_CFG_FLAG_BOOTP`)
If set, the device obtains its configuration from a BOOTP server.

Enable DHCP (`IP_CFG_FLAG_DHCP`)
If set, the device obtains its configuration from a DHCP server.

Please note, that there exists a fallback procedure between the different configuration methods, if more than one is enabled in the `usFlags` variable. If enabled, the device will first try to configure using DHCP. If DHCP configuration fails, the device will fallback to BOOTP, if this is enabled. In case of an BOOTP failure, the values found in the `ulIpAddr`, `ulNetMask`, and `ulGateway` variables will be used, if enabled in `usFlags`. If none of the configuration mechanisms succeeds, the device will report an error. For a description of the timing implied by the different IP configuration mechanisms, please refer to section 3 ‘Start-up of the Device’.

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

Ethernet address available (IP_CFG_FLAG_ETHERNET_ADDR)
If set, the abEthernetAddr area will be evaluated.

Auto-Detect Interface (IP_CFG_FLAG_AUTO_DETECT)
If set, the device will auto-detect the connected interface. The parameter 'Interface Selection' will be ignored in this case.

In auto-detetct mode the Twisted-Pair interface will automatically be enabled, when valid link pulses are detected on this interface. Else the AUI interface will be used.

Please note, that the interface auto-detection does not work reliably on 10 MBit devices, if they are connected to an auto-negotiation hub or switch. In this case the interface selection must be done manually (see 'Interface Selection' below).

Interface Selection (IP_CFG_FLAG_INTF_TP)
If set, the Twisted Pair interface will be selected, else the device will communicate using the AUI. This parameter will not be in effect, when interface auto-detection is active.

Auto-Negotiation (IP_CFG_FLAG_AUTO_NEGOTIATE)
If set, the device will auto-negotiate link parameters with the remote hub or switch. This flag will override the IP_CFG_FLAG_FULL_DUPLEX flag and the IP_CFG_FLAG_-SPEED_100MBIT flag.

Duplex Operation (IP_CFG_FLAG_FULL_DUPLEX)
If set, full-duplex operation will be enabled. The device will operate in half-duplex mode, if this parameter is set to zero. This parameter will not be in effect, when auto-negotiation is active.

Speed Selection (IP_CFG_FLAG_SPEED_100MBIT)
If set, the device will operate at 100 MBit/s, else at 10 MBit/s. This parameter will not be in effect, when auto-negotiation is active.

- **Parameter ulIpAddr**

The devices IP address can be configured using the ulIpAddr variable. Additionally, the IP_CFG_FLAG_IP_ADDR flag must be set in usFlags.

- **Parameter ulNetMask**

The ulNetMask variable holds the Netmask for the subnet the device is connected to. Additionally, the IP_CFG_FLAG_NET_MASK flag must be set in usFlags.

- **Parameter ulGateway**

The ulGateway variable stores the IP address of the default gateway. Additionally, the IP_CFG_FLAG_GATEWAY flag must be set in usFlags.

- **Parameter abEthernetAddr**

The abEthernet area can be used to overwrite the device's default Ethernet address (MAC address). Additionally, the IP_CFG_FLAG_ETHERNET_ADDR flag must be set in usFlags.

2.2 Sending Configuration Parameters Using Messages

In addition to the mechanism described above it is possible to send configuration command messages to the IP task. The messages are explained in detail in section 5.1.2 'Providing Configuration Data – IP_CMD_SET_CONFIG' of this manual.

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

3 Start-up of the Device

After power on or reset the device performs a start-up initialization. During this phase several steps are taken to bring the device from uninitialized state to operation.

First, the hardware will be self-tested and the internal operating system will be started. When this step is finished the device will set the READY bit in the HostFlags cell of the dual-port memory. Please provide a timeout of 2 seconds to your application specific reset function or the DevReset function call. If the READY bit is not set to 1 after this time the device can be assumed to be defective.

During the second step the initialization of the devices tasks will be completed. Initially, both the IP task and the TCP_UDP task will report an error code TASK_NOT_INITIALIZED (200), which indicates that the task is still initializing. You'll find the tasks error code displayed in the dual-port memory in the TaskCondition variable of the respective task. This error code will change to TASK_F_OK (0) once the task has finished the initialization successfully. In case of an initialization error an error code which is different from TASK_NOT_INITIALIZED will be issued. If all tasks in the system are successfully initialized at the end of step 2 the RUN bit of HostFlags will be set.

The task initialization can take up to 209 seconds dependent on the current configuration. The actual time is mainly determined by the DHCP and BOOTP parameters. The DHCP configuration mechanism requires up to 136 seconds and BOOTP requires up to 68 seconds to decide whether it succeeded or failed. If the IP address is configured manually only the check for duplicate IP address assignment will be performed which takes about 3 seconds.

According to the fallback procedure described in section 2.1.3 'Description of the Protocol Parameters' all three IP address configuration mechanisms can be combined. Please look at the table below for a list of all combinations.

IP Configuration Mechanism			Maximum Time Required (seconds)
DHCP	BOOTP	Manually	
		x	3
	x		68
	x	x	71
x			136
x		x	139
x	x		204
x	x	x	209

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

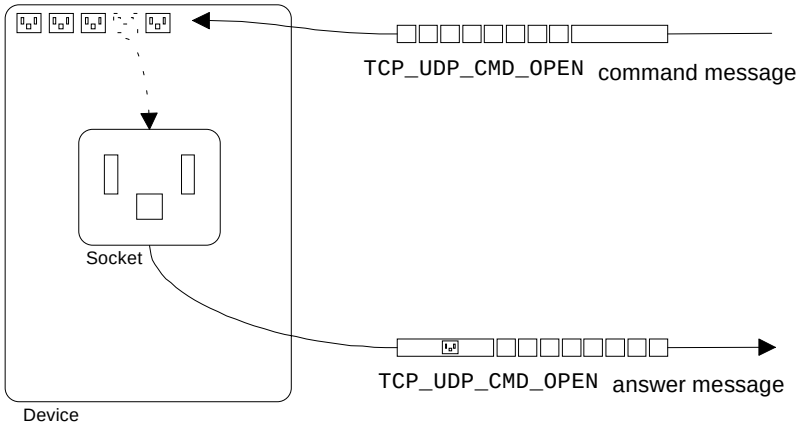
4 Sockets-based Programming Model

The TCP_UDP task deploys a sockets-based approach to the handling of TCP and UDP communication. The socket model implemented in the device is loosely related to the well known BSD sockets or WinSock programming model. However, only the basic features known from these models are implemented in order to simplify the communication and to adapt to the restrictions of an embedded device. The main difference is the message-based communication provided by the device as opposed to the function calls implemented by BSD sockets or WinSock libraries. The function calls are represented by command message and receive message pairs in the TCP_UDP task. You'll find a detailed description of all messages supported in section 5 'Message Interface'.

In order to get an impression of how to make use of the socket communication and how to handle the messages involved some examples will be shown below.

4.1 TCP Client Example

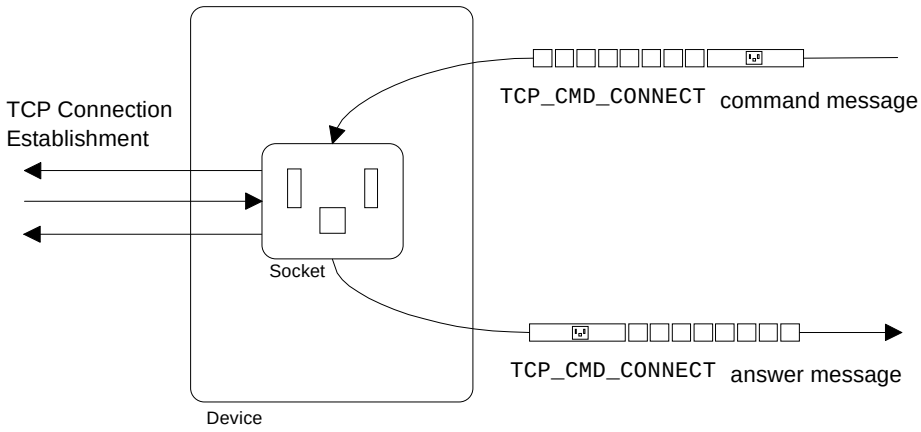
The TCP client example illustrates the message handling required by a simple TCP client application. This application establishes a TCP connection to a remote server, and it sends some request data. The client then reads the response data supplied by the server followed by a shutdown of the connection.



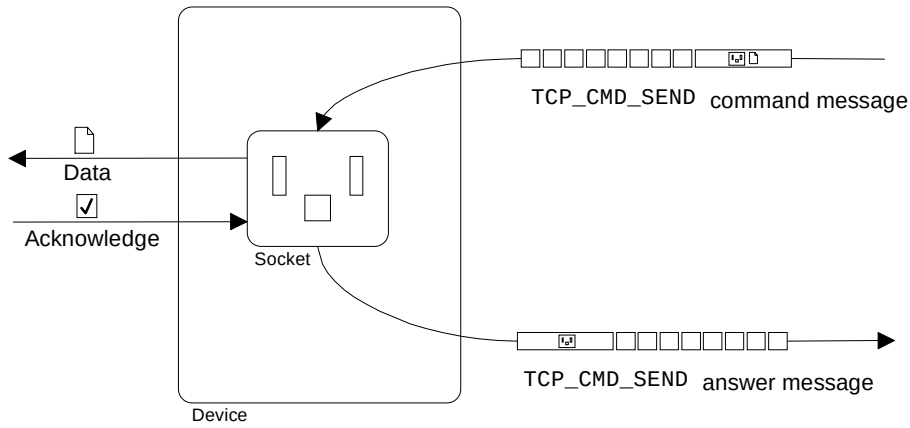
The first step of any socket-based communication is to open a socket of the desired protocol type. Therefore a TCP_UDP_CMD_OPEN command message should be sent to the device. The device will return an answer message containing the handle of the socket just opened.

When the answer message is returned the TCP client application can proceed to the next step which establishes a connection to the TCP server. This will be accomplished by sending a TCP_CMD_CONNECT command message to the device.

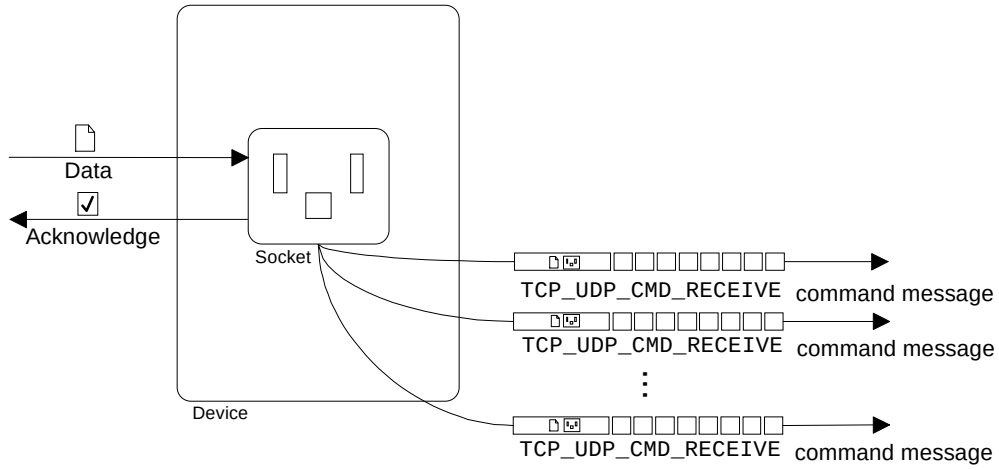
Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E



The device then contacts the server owning the IP address given in the TCP_CMD_CONNECT message. If the connection could be established an answer message reporting success will be returned to the application. In case the IP address cannot be found or the server refuses the connection the answer message will contain an error code indicating the reason of failure.



Once the connection is established the application can then start sending its request data using a TCP_CMD_SEND command message. The device forwards the data to the server, and waits for the server's TCP/IP stack to acknowledge it. When the acknowledgement arrives the device returns an TCP_CMD_SEND answer message to the application.

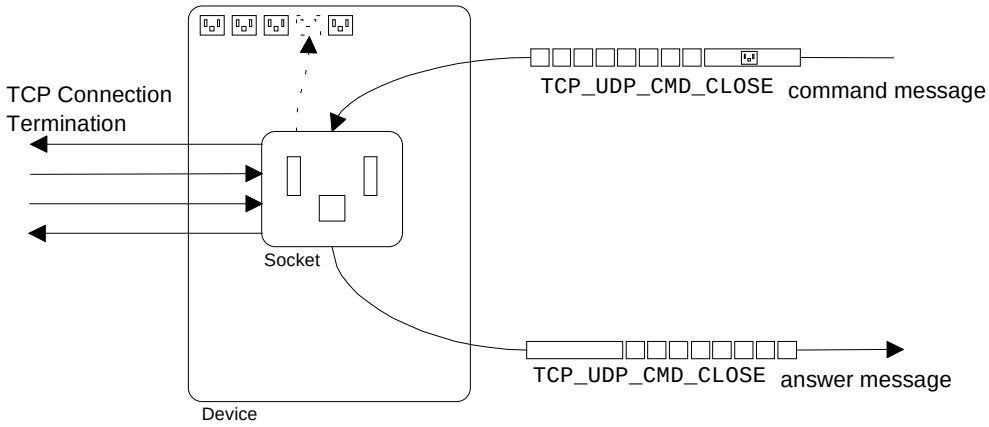


When the server has finished processing, it sends its response data through the TCP

connection. The device receives the data from the network, and sends it to the application using TCP_UDP_CMD_RECEIVE messages. There will be as many TCP_-UDP_CMD_RECEIVE messages as required to transport the server's data.

Please note, that the device sends TCP_UDP_CMD_RECEIVE messages automatically, whenever data is received from the network. There is no explicit receive request message, because the application is assumed to be ready to handle received data once it has established a connection. Thus, the application should check for available messages in the device on a regular basis.

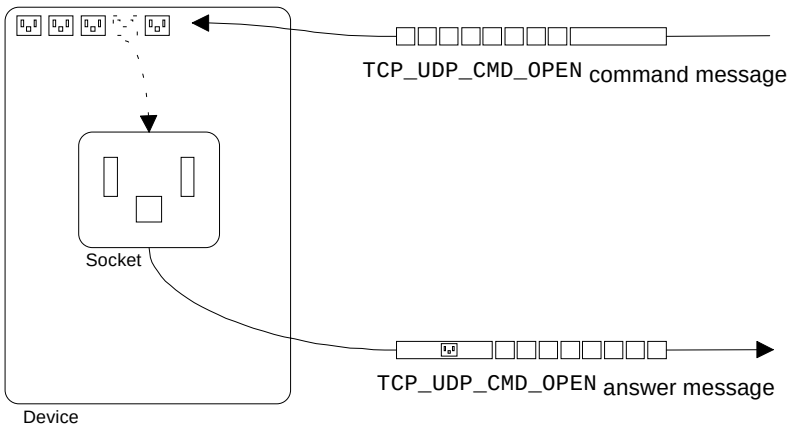
Once the application has received all response data from the server it terminates the TCP connection by sending a TCP_UDP_CMD_CLOSE command message.



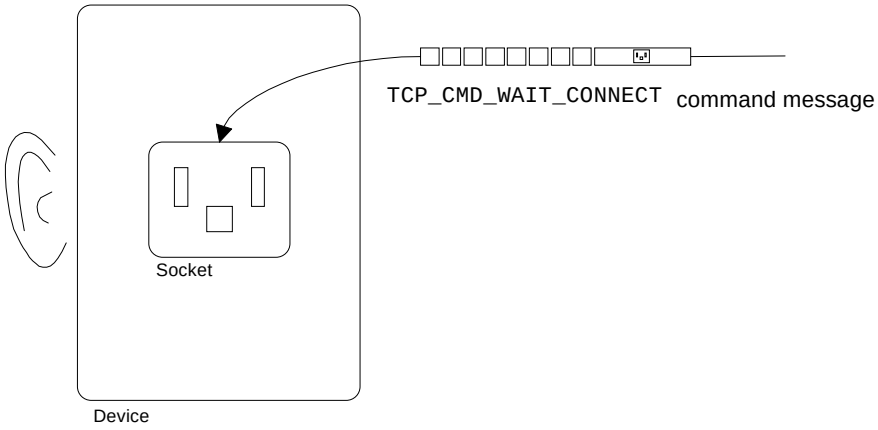
When the connection is terminated the device closes the socket, and a TCP_UDP_CMD_CLOSE answer message is returned to the application.

4.2 TCP Server Example

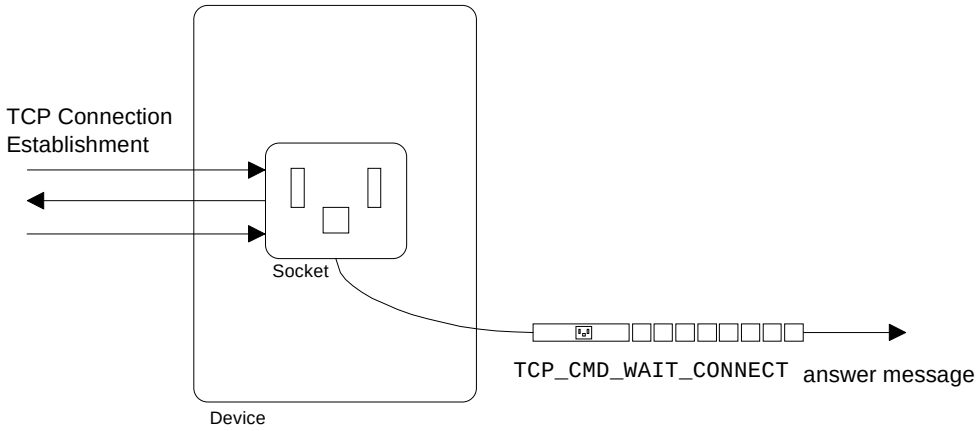
The example shown in this section will enlighten the operation of a very simple TCP server application. This server waits for an incoming request from the network. When the request is received some response data are sent, and the connection is closed afterwards.



A socket must be opened before any TCP communication on the network can start. The server application sends a TCP_UDP_CMD_OPEN message to the device, and waits for the corresponding answer message which returns a socket handle.

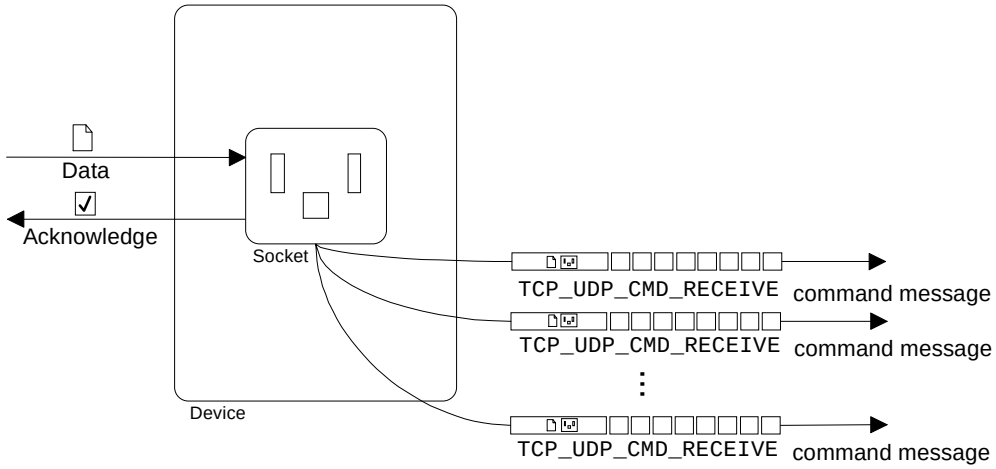


When a socket handle could be obtained, the server application puts this socket into listening state by sending a TCP_CMD_WAIT_CONNECT message. The device defers the answer message to this command as long as the socket is waiting for an incoming connection.

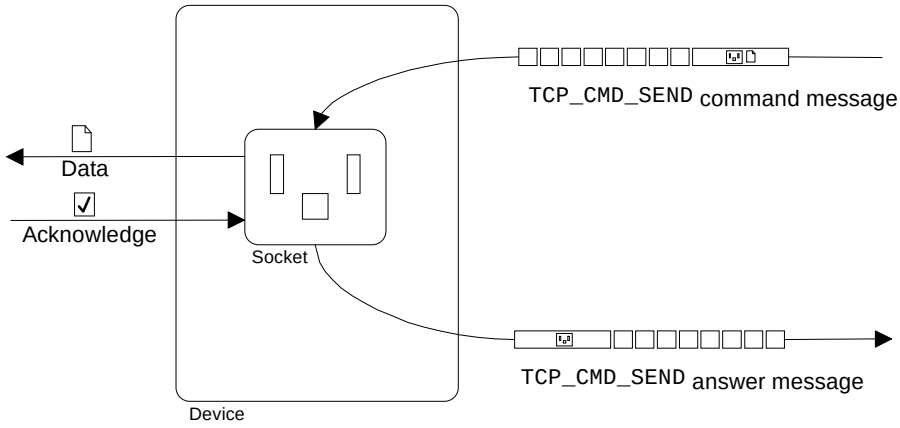


Once a connection request is detected, it is processed, and the socket is no longer listening. The result of the connection establishment is returned in the TCP_CMD_WAIT_CONNECT answer message then. If successful, the socket will be connected to the client that initiated the connection.

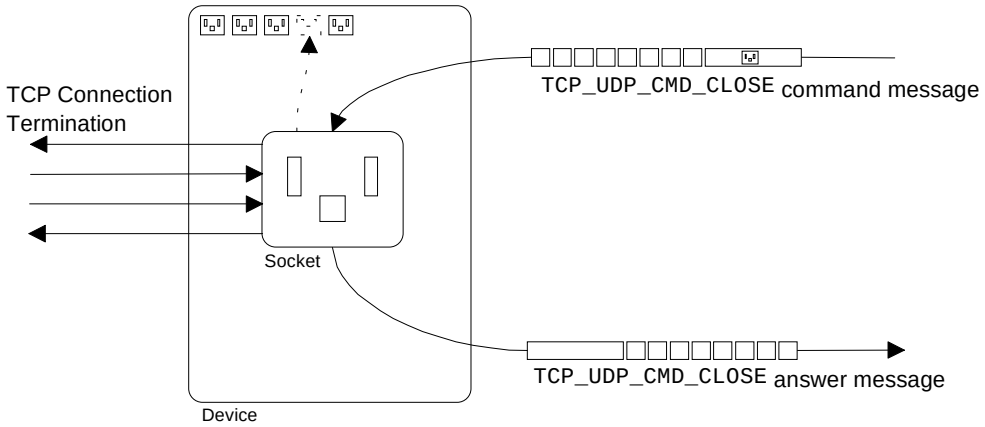
Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E



The server then awaits the client's request data, which is transferred to the server application using TCP_UDP_CMD_RECEIVE messages.



After processing the request the server sends response data back to the client by means of TCP_CMD_SEND messages.

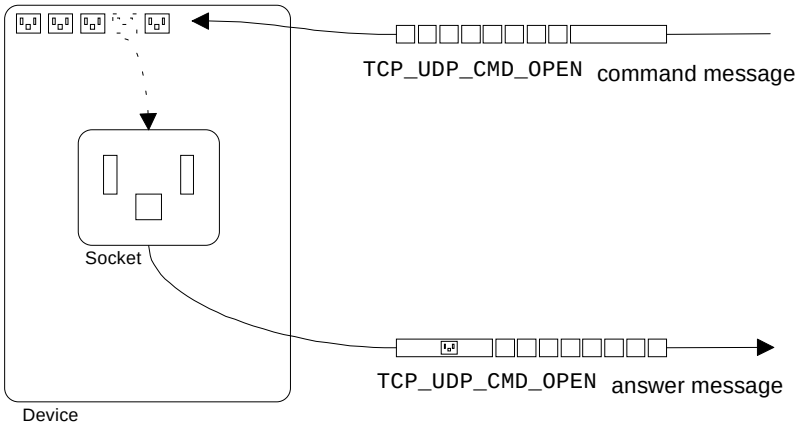


Once the data is transferred the server sends a TCP_UDP_CMD_CLOSE message to the device in order to terminate the TCP connection and to close the socket.

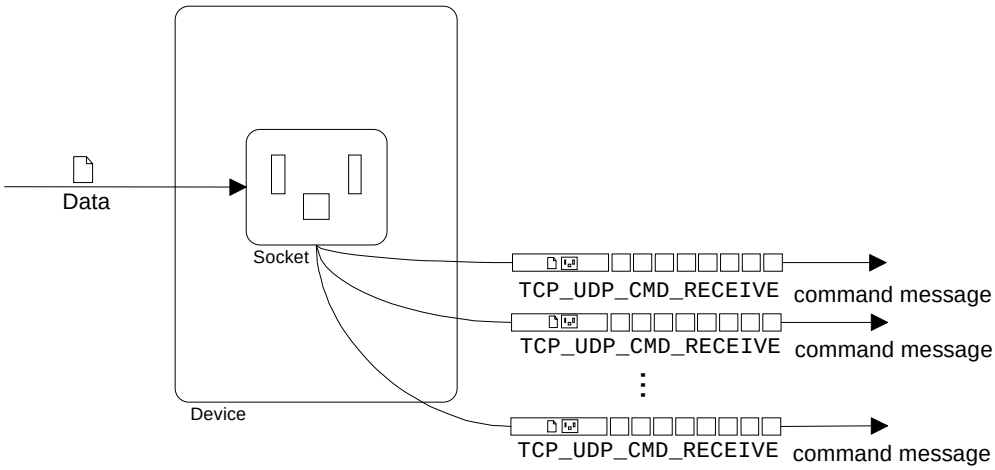
4.3 UDP Communication Example

UDP is a very simple datagram-orientiented protocol layer, which doesn't guarantee data to be delivered reliably. The conecpt of a connection does not exist within UDP, which leads to a simplified communication model compared to TCP. As a result the number of messages that need to be exchanged between application and device is reduced, too.

The example shown below illustrates which messages need to be handled by an application that communicates using UDP.

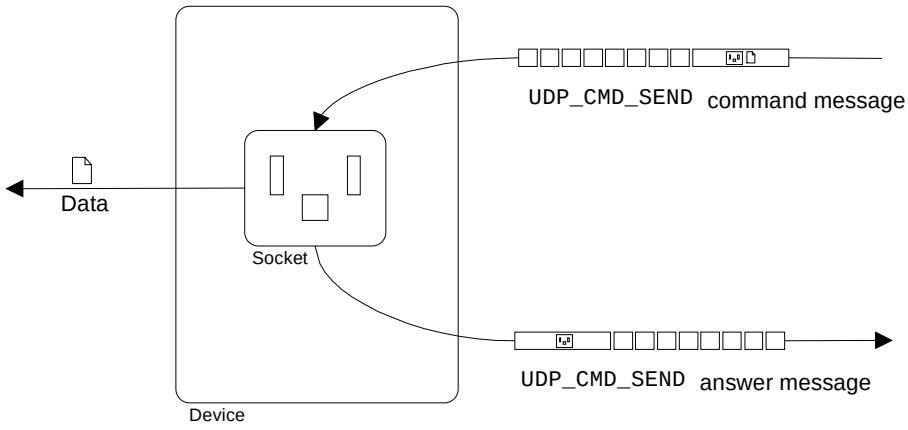


Most simply, only a UDP socket needs to be opened in order to enable an application to receive UDP data. The application sends a TCP_UDP_CMD_OPEN command message to the device, and waits for the corresponding answer message to be returned.

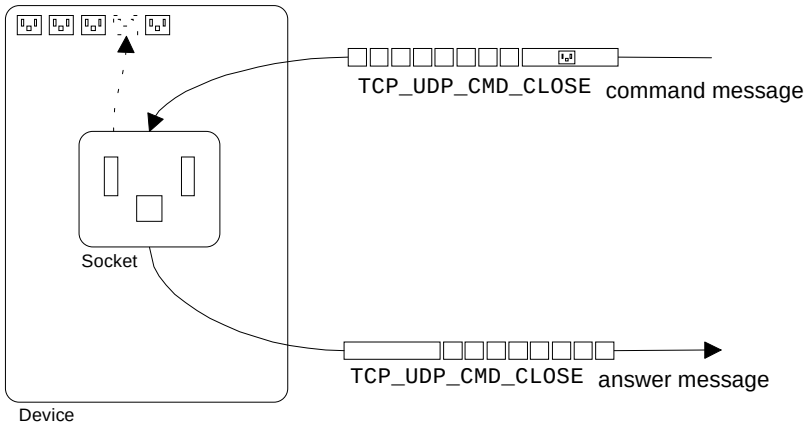


Once the socket is opened all incoming UDP data for this socket will be sent to the application by means of TCP_UDP_CMD_RECEIVE messages.

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E



The application processes the data, and sends its reponse data using UDP_CMD_SEND messages.



When the application wants to stop UDP communication, it sends a TCP_UDP_-CMD_CLOSE command message in order to close the corresponding socket.

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

5 Message Interface

The messages described in the following sections can be exchanged with the device using its mailboxes. You'll find a description of the message transfer mechanism in general in the 'Toolkit Manual - General Definitions'.

To put a message to, respectively to get a message from the device use the driver functions `DevPutMessage()` or `DevGetMessage()`. With direct access to the dual-port memory the application puts a message into the `DevMailbox`, respectively reads a message from the `HostMailbox` using the mechanism described in the Toolkit Manual 'General Definitions'.

5.1 The IP Task

5.1.1 Configuration of the IP Task

Normally the configuration will be stored into the non-volatile flash memory by the configuration tool SyCon. The device reads this data block during its start-up initialization and performs consistency checks.

The user application can add, change or delete parameters during run-time. These parameters will be kept in the RAM area of the device. For this purpose the message `IP_SET_CONFIG` can be used.

NOTE: Parameter sets stored in the RAM area will be lost if the board is powered down or if a cold-start or warm-start is performed.

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

5.1.2 Providing Configuration Data – IP_CMD_SET_CONFIG

Using the IP_CMD_SET_CONFIG message the IP task can be provided with new configuration parameters. If any sockets are open when the IP_CMD_SET_CONFIG message is recevided by the device it will send a TCP_CMD_SHUTDOWN message to the owner of the socket. Please refer to the section 5.2.11 'Shutdown of the Device – TCP_UDP_CMD_SHUTDOWN' to learn more about the handling of the TCP_CMD_SHUTDOWN message.

After sending the configuration message, the IP task and the TCP task will be re-started with the new configuration in effect. The timing considerations presented in section 3 'Start-up of the Device' apply to the IP_CMD_SET_CONFIG message, too.

	Command Message			
	Parameter	Type	Value	Description
Message Header	msg.rx	USIGN8	7	Identification of Receiver IP task
	msg.tx	USIGN8	16	Identification of Transmitter User application
	msg.ln	USIGN8	20	Message Length Length in octets
	msg.nr	USIGN8	0 ... 255	Message Identification Unique number
	msg.a	USIGN8	0	Reply Identification Not a reply
	msg.f	USIGN8	0	Error Number No error
	msg.b	USIGN8	32	Command Identification IP_CMD_SET_CONFIG command
	msg.e	USIGN8	0	Extension Standard
Message Data	msg.d[0]	USIGN16	see below	Flags Flags
	msg.d[2]	USIGN32		IP Address IP address of device
	msg.d[6]	USIGN32		Netmask Netmask of local subnet
	msg.d[10]	USIGN32		Gateway IP address of default gateway
	msg.d[14]	USIGN8 [6]		Ethernet Address Ethernet address of the device

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

- **Parameter usFlags**

The `usFlags` variable holds bit-oriented flags according to the following table:

Flags (Bit 0 ... 7)							
D7	D6	D5	D4	D3	D2	D1	D0
							IP address available
						Netmask available	
					Gateway available		
				Enable BOOTP			
			Enable DHCP				
		Set Ethernet address					
Reserved, set to zero							

D15	D14	D13	D12	D11	D10	D9	D8
							Auto-Detect Interface
						Interface Selection	
					Auto-Negotiation		
				Duplex Operation			
			Speed Selection				
Reserved, set to zero							

Please refer to section 2.1.3 ‘Description of the Protocol Parameters’ for a detailed description of the remaining flag bits.

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

The IP task will answer the `IP_CMD_SET_CONFIG` command message with the following answer message:

	Answer Message			
	Parameter	Type	Value	Description
Message Header	msg.rx	USIGN8		Identification of Receiver
			16	User application
	msg.tx	USIGN8		Identification of Transmitter
			7	IP task
	msg.ln	USIGN8		Message Length
			0	Length in octets
	msg.nr	USIGN8		Message Identification
			0 ... 255	Unique number
	msg.a	USIGN8		Reply Identification
			32	IP_CMD_SET_CONFIG
	msg.f	USIGN8		Error Number
			0	No error
			52	No Ethernet address available
			56	Invalid subnet mask
			57	Invalid gateway IP address
			60	No IP address available
			61	Driver failed to initialize
			62	No source for an IP address specified
			151	Invalid message length
			152	Unknown message command
			200	Task not initialized
	msg.b	USIGN8		Command Identification
			0	No command
	msg.e	USIGN8		Extension
			0	Standard

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

5.1.3 Obtaining Configuration Data – IP_CMD_GET_CONFIG

The IP_CMD_GET_CONFIG command can be used to obtain the current configuration from the IP task. Parameters returned include IP address, netmask, IP address of default gateway, and several flags.

	Command Message			
	Parameter	Type	Value	Description
Message Header	msg.rx	USIGN8	7	Identification of Receiver IP task
	msg.tx	USIGN8	16	Identification of Transmitter User application
	msg.ln	USIGN8	0	Message Length Length in octets
	msg.nr	USIGN8	0 ... 255	Message Identification Unique number
	msg.a	USIGN8	0	Reply Identification Not a reply
	msg.f	USIGN8	0	Error Number No error
	msg.b	USIGN8	33	Command Identification IP_CMD_GET_CONFIG command
	msg.e	USIGN8	0	Extension Standard

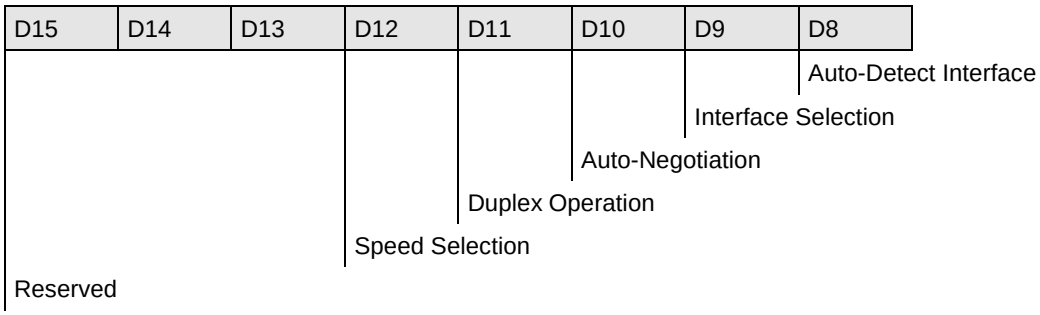
Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

The IP task will respond with the following message:

	Answer Message			
	Parameter	Type	Value	Description
Message Header	msg.rx	USIGN8	16	Identification of Receiver User application
	msg.tx	USIGN8	7	Identification of Transmitter IP task
	msg.ln	USIGN8	20	Message Length Length in octets
	msg.nr	USIGN8	0 ... 255	Message Identification Unique number
	msg.a	USIGN8	33	Reply Identification IP_CMD_GET_CONFIG
	msg.f	USIGN8	0 151 152 200	Error Number No error Invalid message length Unknown message command Task not initialized
	msg.b	USIGN8	0	Command Identification No command
	msg.e	USIGN8	0	Extension Standard
Message Data	msg.d[0]	USIGN16	see below	Flags Flags
	msg.d[2]	USIGN32		IP Address IP address of device
	msg.d[6]	USIGN32		Netmask Netmask of local subnet
	msg.d[10]	USIGN32		Gateway IP address of default gateway
	msg.d[14]	USIGN8 [6]		Ethernet Address Ethernet address of the device

Flags (Bit 0 ... 7)							
D7	D6	D5	D4	D3	D2	D1	D0

D0: IP address available
D1: Netmask available
D2: Gateway available
D3: BOOTP enabled
D4: DHCP enabled
D5 ... D7: Reserved



Please refer to the section 2.1.3 ‘Description of the Protocol Parameters’ for a detailed description of the remaining flag bits.

5.1.4 Setting IP Parameters – IP_CMD_SET_PARAM

Some parameters of the IP stack can be modified during run-time. Using the IP_CMD_SET_PARAM command message entries in the devices ARP cache can be added or removed. For a definition of the corresponding message, see the table below.

	Command Message			
	Parameter	Type	Value	Description
Message Header	msg.rx	USIGN8	7	Identification of Receiver IP task
	msg.tx	USIGN8	16	Identification of Transmitter User application
	msg.ln	USIGN8	see below	Message Length Length in octets
	msg.nr	USIGN8	0 ... 255	Message Identification Unique number
	msg.a	USIGN8	0	Reply Identification No reply
	msg.f	USIGN8	0	Error Number No error
	msg.b	USIGN8	34	Command Identification IP_CMD_SET_PARAM command
	msg.e	USIGN8	0	Extension Standard
Message Data	msg.d[0]	USIGN8	see below	Mode Type of parameter to configure
	msg.d[1]	USIGN8	0	Reserved Reserved for future use
	msg.d[2 ...]	see below	see below	Data Parameter data to set

Mode	Description	Data	Data Type	msg.ln
1	IP_PRM_ADD_ARP_ENTRY Add static entry to ARP cache	IP address MAC address	USIGN32 USIGN8[6]	12
2	IP_PRM_DEL_ARP_ENTRY Delete entry from ARP cache	IP address MAC address	USIGN32 USIGN8[6]	12
3	IP_PRM_DEL_ARP_ENTRY_IP Delete entry from ARP cache	IP address	USIGN32	6
4	IP_PRM_DEL_ARP_ENTRY_MAC Delete all entries with specified MAC address from ARP cache	MAC address	USIGN8[6]	8

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

The IP task responds with the following message:

	Answer Message			
	Parameter	Type	Value	Description
Message Header	msg.rx	USIGN8	16	Identification of Receiver User application
	msg.tx	USIGN8	7	Identification of Transmitter IP task
	msg.ln	USIGN8	2	Message Length Length in octets
	msg.nr	USIGN8	0 ... 255	Message Identification Unique number
	msg.a	USIGN8	34	Reply Identification IP_CMD_SET_PARAM
	msg.f	USIGN8	0 124 130 131 134 151 152 200	Error Number No error Invalid Ethernet address Invalid mode parameter ARP cache is full ARP entry not found in ARP cache Invalid message length Unknown message command Task not initialized
	msg.b	USIGN8	0	Command Identification Not a command
	msg.e	USIGN8	0	Extension Standard
Message Data	msg.d[0]	USIGN8	see below	Mode Type of parameter
	msg.d[1]	USIGN8	0	Reserved Reserved for future use

Mode	Description
1	IP_PRM_ADD_ARP_ENTRY Add static entry to ARP cache
2	IP_PRM_DEL_ARP_ENTRY Delete entry from ARP cache
3	IP_PRM_DEL_ARP_ENTRY_IP Delete entry from ARP cache
4	IP_PRM_DEL_ARP_ENTRY_MAC Delete all entries with specified MAC address from ARP cache

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

5.1.5 Obtaining IP Parameters – `IP_CMD_GET_PARAM`

The `IP_CMD_GET_PARAM` command message provides an opportunity to obtain current parameters from the IP task. Currently access to the IP stacks ARP cache is implemented.

	Command Message			
	Parameter	Type	Value	Description
Message Header	msg.rx	USIGN8	7	Identification of Receiver IP task
	msg.tx	USIGN8	16	Identification of Transmitter User application
	msg.ln	USIGN8	see below	Message Length Length in octets
	msg.nr	USIGN8	0 ... 255	Message Identification Unique number
	msg.a	USIGN8	0	Reply Identification No reply
	msg.f	USIGN8	0	Error Number No error
	msg.b	USIGN8	35	Command Identification IP_CMD_GET_PARAM command
	msg.e	USIGN8	0	Extension Standard
Message Data	msg.d[0]	USIGN8	see below	Mode Type of parameter to be obtained
	msg.d[1]	USIGN8	0	Reserved Reserved for future use
	msg.d[2 ...]	see below	see below	Data Additional data required by parameter

Mode	Description	Data	Data Type	msg.ln
1	IP_PRM_GET_ARP_ENTRY_INDEX Get ARP entry at specified cache index	ARP cache index	USIGN16	4
2	IP_PRM_GET_ARP_ENTRY_IP Get entry with specified IP address from ARP cache	IP address	USIGN32	6
3	IP_PRM_GET_ARP_ENTRY_MAC Get first entry with specified MAC address from ARP cache	MAC address	USIGN8[6]	8

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

The IP task responds with the following message:

	Answer Message			
	Parameter	Type	Value	Description
Message Header	msg.rx	USIGN8	16	Identification of Receiver User application
	msg.tx	USIGN8	7	Identification of Transmitter IP task
	msg.ln	USIGN8	see below	Message Length Length in octets
	msg.nr	USIGN8	0 ... 255	Message Identification Unique number
	msg.a	USIGN8	35	Reply Identification IP_CMD_GET_PARAM answer
	msg.f	USIGN8	0 130 134 151 152 200	Error Number No error Invalid mode parameter ARP entry not found in ARP cache Invalid message length Unknown message command Task not initialized
	msg.b	USIGN8	0	Command Identification No command
	msg.e	USIGN8	0	Extension Standard
Message Data	msg.d[0]	USIGN8	see below	Mode Type of parameter
	msg.d[1]	USIGN8	0	Reserved Reserved for future use
	msg.d[2 ...]	see below	see below	Data Result data

Mode	Description	Data	Data Type	msg.ln
1	IP_PRM_GET_ARP_ ENTRY_INDEX Entry from ARP cache	IP address Ethernet address	USIGN32 USIGN8[6]	12
2	IP_PRM_GET_ARP_ ENTRY_IP Entry from ARP cache	IP address Ethernet address	USIGN32 USIGN8[6]	12
3	IP_PRM_GET_ARP_ ENTRY_MAC Entry from ARP cache	IP address Ethernet address	USIGN32 USIGN8[6]	12

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

5.1.6 Obtaining TCP/IP Stack Capabilities – `IP_CMD_GET_OPTION`

The `IP_CMD_GET_OPTION` message instructs the IP stack to return information about supported protocols to the application.

	Command Message			
	Parameter	Type	Value	Description
Message Header	msg.rx	USIGN8	7	Identification of Receiver IP task
	msg.tx	USIGN8	16	Identification of Transmitter User application
	msg.ln	USIGN8	0	Message Length Length in octets
	msg.nr	USIGN8	0 ... 255	Message Identification Unique number
	msg.a	USIGN8	0	Reply Identification No reply
	msg.f	USIGN8	0	Error Number No error
	msg.b	USIGN8	37	Command Identification IP_CMD_GET_OPTION command
	msg.e	USIGN8	0	Extension Standard

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

The IP task responds with the following message:

	Answer Message			
	Parameter	Type	Value	Description
Message Header	msg.rx	USIGN8	16	Identification of Receiver User application
	msg.tx	USIGN8	7	Identification of Transmitter IP task
	msg.ln	USIGN8	2	Message Length Length in octets
	msg.nr	USIGN8	0 ... 255	Message Identification Unique number
	msg.a	USIGN8	37	Reply Identification IP_CMD_GET_OPTION answer
	msg.f	USIGN8	0 151 152 200	Error Number No error Invalid message length Unknown message command Task not initialized
	msg.b	USIGN8	0	Command Identification Not a command
	msg.e	USIGN8	0	Extension Standard
Message Data	msg.d[0]	USIGN16	see below	Options Options supported by TCP/IP stack

Options supported by TCP/IP stack:

Options (Bit 0 ... 7)							
D7	D6	D5	D4	D3	D2	D1	D0
Reserved			IP Multicast	DHCP	BOOTP	UDP	TCP

Options (Bit 8 ... 15)							
D15	D14	D13	D12	D11	D10	D9	D8
Reserved							

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

5.1.7 Sending a Ping – `IP_CMD_PING`

The `IP_CMD_PING` command can be used to send an ICMP Echo Request packet (Internet Control Message Protocol) to the specified IP address. The target IP stack should answer with an ICMP Echo Reply packet. This command is similar to the commonly known “ping” program.

	Command Message			
	Parameter	Type	Value	Description
Message Header	msg.rx	USIGN8	7	Identification of Receiver IP task
	msg.tx	USIGN8	16	Identification of Transmitter User application
	msg.ln	USIGN8	8	Message Length Length in octets
	msg.nr	USIGN8	0 ... 255	Message Identification Unique number
	msg.a	USIGN8	0	Reply Identification No reply
	msg.f	USIGN8	0	Error Number No error
	msg.b	USIGN8	38	Command Identification IP_CMD_PING command
	msg.e	USIGN8	0	Extension Standard
Message Data	msg.d[0]	USIGN32		IP address IP address of the target system
	msg.d[4]	USIGN32	see below	Timeout Timeout for request

Timeout	Description
0	Return immediately, don’t wait for an ICMP Echo Reply packet
1 ... 2147483647	Wait up to specified time for return of an ICMP Echo Reply packet (time in milliseconds).

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

The IP task responds with the following message:

	Answer Message			
	Parameter	Type	Value	Description
Message Header	msg.rx	USIGN8	16	Identification of Receiver User application
	msg.tx	USIGN8	7	Identification of Transmitter IP task
	msg.ln	USIGN8	4	Message Length Length in octets
	msg.nr	USIGN8	0 ... 255	Message Identification Unique number
	msg.a	USIGN8	38	Reply Identification IP_CMD_PING answer
	msg.f	USIGN8	0 110 111 118 151 152 200	Error Number No error Request timed out Invalid timeout parameter Invalid IP address Invalid message length Unknown message command Task not initialized
	msg.b	USIGN8	0	Command Identification No command
	msg.e	USIGN8	0	Extension Standard
Message Data	msg.d[0]	USIGN32		Response Time Response time of the Ping answer (milliseconds)

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

5.2 The TCP/UDP Task

5.2.1 Opening a Socket – TCP_UDP_CMD_OPEN

The TCP_UDP_CMD_OPEN command can be used by the application to obtain a handle for a socket of the specified protocol type. This is always the first step to be taken for any socket-based communication. The newly created socket will be bound to the specified local IP address and local port number.

When working with an UDP socket the application can start sending data right away (UDP_CMD_SEND), once the socket is successfully opened. Some more steps are required for TCP sockets: A connection must be established using either the TCP_CMD_WAIT_CONNECT or TCP_CMD_CONNECT command messages before being able transfer data.

	Command Message			
	Parameter	Type	Value	Description
Message Header	msg.rx	USIGN8	6	Identification of Receiver TCP_UDP task
	msg.tx	USIGN8	16	Identification of Transmitter User application
	msg.ln	USIGN8	8	Message Length Length in octets
	msg.nr	USIGN8	0 ... 255	Message Identification Unique number
	msg.a	USIGN8	0	Reply Identification No reply
	msg.f	USIGN8	0	Error Number No error
	msg.b	USIGN8	48	Command Identification TCP_UDP_CMD_OPEN command
	msg.e	USIGN8	0	Extension Standard
Message Data	msg.d[0]	USIGN32	see below	Local IP address Local IP address to use with socket
	msg.d[4]	USIGN16	see below	Local Port Number Local port number to use with socket
	msg.d[6]	USIGN8	0	Reserved Reserved for future use
	msg.d[7]	USIGN8	see below	Protocol Type Protocol type to use

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

Local IP address	Description
0 (0.0.0.0)	Bind socket to currently configured IP address
unequal 0	Bind socket to specified IP address, must match currently configured IP address

Local Port Number	Description
0	Bind socket to next available port in range 1024 ... 65535
1 ... 65535	Bind socket to specified port

Protocol Type	Description
1	TCP_PROTO_TCP Transmission Control Protocol (TCP)
2	TCP_PROTO_UDP User Datagram Protocol (UDP)

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

The TCP_UDP task responds with the following message:

	Answer Message			
	Parameter	Type	Value	Description
Message Header	msg.rx	USIGN8	16	Identification of Receiver User application
	msg.tx	USIGN8	6	Identification of Transmitter TCP_UDP task
	msg.ln	USIGN8	1	Message Length Length in octets
	msg.nr	USIGN8	0 ... 255	Message Identification Unique number
	msg.a	USIGN8	48	Reply Identification TCP_UDP_CMD_OPEN
	msg.f	USIGN8	0 118 119 122 123 151 152 200	Error Number No error Invalid local IP address Invalid local port Protocol unknown No sockets available Invalid message length Unknown message command Task not initialized
	msg.b	USIGN8	0	Command Identification No command
	msg.e	USIGN8	0	Extension Standard
Message Data	msg.d[0]	USIGN8	0 ... 255	Socket handle Socket handle

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

5.2.2 Closing a Socket – TCP_UDP_CMD_CLOSE

The TCP_UDP_CMD_CLOSE command message works for both TCP and UDP sockets. It will close a currently active socket, and will invalidate its socket handle. If the socket was connected to a remote TCP communication partner the connection will be terminated.

The command message expects a socket handle and a timeout value to be provided. The timeout parameter applies to TCP sockets, and it will work the following way: The device will send its connection termination request, and will then wait up to the specified time for the remote TCP stack to close the connection, too (graceful close). If the timeout is exceeded a TCP connection reset will be forced (hard close). A connection reset will always be performed if the timeout value is set to -1 (0FFFFFFFFh).

NOTE: Please note, that the TCP_UDP_CMD_CLOSE answer message will be delayed for 2 seconds for TCP sockets, if the local TCP stack was the one, which initiated the TCP connection termination (active close).

UDP sockets will always be closed immediately. The timeout value must be set to zero in this case.

	Command Message			
	Parameter	Type	Value	Description
Message Header	msg.rx	USIGN8	6	Identification of Receiver TCP_UDP task
	msg.tx	USIGN8	16	Identification of Transmitter User application
	msg.ln	USIGN8	12	Message Length Length in octets
	msg.nr	USIGN8	0 ... 255	Message Identification Unique number
	msg.a	USIGN8	0	Reply Identification Not a reply
	msg.f	USIGN8	0	Error Number No error
	msg.b	USIGN8	49	Command Identification TCP_UDP_CMD_CLOSE command
	msg.e	USIGN8	0	Extension Standard
Message Data	msg.d[0]	USIGN32	0	Reserved Reserved for future use
	msg.d[4]	USIGN16	0	Reserved Reserved for future use
	msg.d[6]	USIGN8	0 ... 255	Socket Handle Socket handle to close
	msg.d[7]	USIGN8	0	Reserved Reserved for future use
	msg.d[8]	USIGN32	see below	Timeout Timeout

Timeout	Description

0	Default timeout (13000 milliseconds)
1 ... 2147483647	Wait up to specified time for completion of the close operation (graceful close), if timeout is exceeded perform connection reset. (timeout in milliseconds) **Applicable to TCP sockets only**
0FFFFFFFFh	Close socket immediately (connection reset) **Applicable to TCP sockets only**

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

The TCP_UDP task responds with the following message:

	Answer Message			
	Parameter	Type	Value	Description
Message Header	msg.rx	USIGN8	16	Identification of Receiver User application
	msg.tx	USIGN8	6	Identification of Transmitter TCP_UDP task
	msg.ln	USIGN8	8	Message Length Length in octets
	msg.nr	USIGN8	0 ... 255	Message Identification Unique number
	msg.a	USIGN8	49	Reply Identification TCP_UDP_CMD_CLOSE
	msg.f	USIGN8	0 110 111 112 115 121 151 152 158 200	Error Number No error Timeout expired Invalid timeout parameter Invalid socket handle Destination unreachable Connection reset Invalid message length Unknown message command Command already running Task not initialized
	msg.b	USIGN8	0	Command Identification No command
	msg.e	USIGN8	0	Extension Standard
Message Data	msg.d[0]	USIGN32	0	Reserved Reserved for future use
	msg.d[4]	USIGN16	0	Reserved Reserved for future use
	msg.d[6]	USIGN8	0 ... 255	Socket handle Handle of socket
	msg.d[7]	USIGN8	0	Reserved Reserved for future use

NOTE: The error codes 110 (Timeout expired), 115 (Destination unreachable), and 121 (Connection reset) rather should be treated as a notification code, because the socket is closed in these cases, too. However, if one of the other error codes is returned, the socket's state is not affected.

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

5.2.3 Closing All Sockets – TCP_UDP_CMD_CLOSE_ALL

Sending the TCP_UDP_CMD_CLOSE_ALL message will instruct the device to close all sockets currently in use by the hosts application program. Thus, all previously obtained socket handles will become invalid.

The timeout value specified in the message will apply to TCP sockets only. UDP sockets will always be closed immediately.

If a timeout value unequal zero is given, this timeout will be valid for all TCP sockets that have to be closed. If the timeout is set to zero, the current timeout for data send operations on the individual TCP socket will be used. TCP connections that time out during the close operation will automatically be aborted with a TCP connection reset.

A possible use of the TCP_UDP_CMD_CLOSE_ALL message is to free all allocated socket resources, once the host application ends communication to the device.

	Command Message			
	Parameter	Type	Value	Description
Message Header	msg.rx	USIGN8	6	Identification of Receiver TCP_UDP task
	msg.tx	USIGN8	16	Identification of Transmitter User application
	msg.ln	USIGN8	12	Message Length Length in octets
	msg.nr	USIGN8	0 ... 255	Message Identification Unique number
	msg.a	USIGN8	0	Reply Identification No reply
	msg.f	USIGN8	0	Error Number No error
	msg.b	USIGN8	56	Command Identification TCP_UDP_CMD_CLOSE_ALL command
	msg.e	USIGN8	0	Extension Standard
Message Data	msg.d[0]	USIGN32	0	Reserved Reserved for future use
	msg.d[4]	USIGN16	0	Reserved Reserved for future use
	msg.d[6]	USIGN8	0	Reserved Reserved for future use
	msg.d[7]	USIGN8	0	Reserved Reserved for future use
	msg.d[8]	USIGN32	see below	Timeout Timeout

Timeout parameter:

Timeout	Description

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

0	Use the individual socket's data send timeout for TCP close operation.
1 ... 2147483647	Timeout value for all TCP close operations (milliseconds)
0FFFFFFFFh	Close TCP sockets immediately (connection reset)

The TCP_UDP task responds with the following message:

	Answer Message			
	Parameter	Type	Value	Description
Message Header	msg.rx	USIGN8	16	Identification of Receiver User Application
	msg.tx	USIGN8	6	Identification of Transmitter TCP_UDP task
	msg.ln	USIGN8	8	Message Length Length in octets
	msg.nr	USIGN8	0 ... 255	Message Identification Unique Number
	msg.a	USIGN8	56	Reply Identification TCP_UDP_CMD_CLOSE_ALL
	msg.f	USIGN8	0 110 111 115 121 151 152 158 200	Error Number No error Timeout expired Invalid timeout parameter Destination unreachable Connection reset Invalid message length Unknown message command Command already running Task not initialized
	msg.b	USIGN8	0	Command Identification No command
	msg.e	USIGN8	0	Extension Standard
Message Data	msg.d[0]	USIGN32	0	Reserved Reserved for future use
	msg.d[4]	USIGN16	0	Reserved Reserved for future use
	msg.d[6]	USIGN8	0	Reserved Reserved for future use
	msg.d[7]	USIGN8	0	Reserved Reserved for future use

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

NOTE: The answer messages returns the result code from the last socket closed in its **msg.f** field. Error codes of 110 (Timeout expired), 115 (Destination unreachable), and 121 (Connection reset) rather should be treated as a notification code, because all sockets were closed in these cases, too. However, if one of the other error codes is returned, the command was rejected immediately and the socket's state is not affected.

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

5.2.4 Waiting for an Incoming TCP Connection – TCP_CMD_WAIT_CONNECT

Using the TCP_CMD_WAIT_CONNECT message a socket can be put into listening state in order to wait for an incoming TCP connection. A TCP server application waiting for connection requests from remote clients may serve as an example here.

The command requires the handle of a previously opened socket and two timeout values to be provided. The send timeout value determines a maximum wait time for future TCP_CMD_SEND commands on this socket. Using the connect timeout parameter a maximum time to wait for incoming connections can be given.

Usually, the TCP_CMD_WAIT_CONNECT command message will not be answered immediately. An immediate answer message will only be returned if invalid message parameters are detected. In the error free case the answer message will be deferred until a connection request comes in. If the **msg.f** field indicates no error, a connection could be created, and the socket is currently in established state. In any other case no connection could be established, and the socket is no longer listening.

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

	Command Message			
	Parameter	Type	Value	Description
Message Header	msg.rx	USIGN8	6	Identification of Receiver TCP_UDP task
	msg.tx	USIGN8	16	Identification of Transmitter User application
	msg.ln	USIGN8	16	Message Length Length in octets
	msg.nr	USIGN8	0 ... 255	Message Identification Unique number
	msg.a	USIGN8	0	Reply Identification No reply
	msg.f	USIGN8	0	Error Number No error
	msg.b	USIGN8	50	Command Identification TCP_CMD_WAIT_CONNECT command
	msg.e	USIGN8	0	Extension Standard
Message Data	msg.d[0]	USIGN32	0	Reserved Reserved for future use
	msg.d[4]	USIGN16	0	Reserved Reserved for future use
	msg.d[6]	USIGN8	0 ... 255	Socket handle Handle of socket to connect
	msg.d[7]	USIGN8	0	Reserved Reserved for future use
	msg.d[8]	USIGN32	see below	Send Timeout Timeout for future send commands
	msg.d[12]	USIGN32	see below	Connect Timeout Timeout for wait connect command

Send Timeout	Description
0	Default timeout (31000 milliseconds)
1 ... 2147483647	Wait up to specified time for transfer of data (time in milliseconds)

Connect Timeout	Description
0	Wait until connection comes in
1 ... 2147483647	Wait up to specified time for incoming connection (time in milliseconds)

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

TCP_CMD_WAIT_CONNECT answer message:

	Answer Message			
	Parameter	Type	Value	Description
Message Header	msg.rx	USIGN8	16	Identification of Receiver User application
	msg.tx	USIGN8	6	Identification of Transmitter TCP_UDP task
	msg.ln	USIGN8	8	Message Length Length in octets
	msg.nr	USIGN8	0 ... 255	Message Identification Unique number
	msg.a	USIGN8	50	Reply Identification TCP_CMD_WAIT_CONNECT
	msg.f	USIGN8	0 110 111 112 113 115 121 151 152 158 200	Error Number No error Timeout expired Invalid timeout parameter Invalid socket handle Not possible in current socket state Destination unreachable Connection reset Invalid message length Unknown message command Command already running Task not initialized
	msg.b	USIGN8	0	Command Identification No command
	msg.e	USIGN8	0	Extension Standard
Message Data	msg.d[0]	USIGN32		IP Address IP address of remote client
	msg.d[4]	USIGN16	0...65535	Port Number Port number of remote client
	msg.d[6]	USIGN8	0 ... 255	Socket Handle Handle of socket for the incoming connection
	msg.d[7]	USIGN8	0	Reserved Reserved for future use

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

5.2.5 Establishing a TCP Connection – TCP_CMD_CONNECT

The TCP_CMD_CONNECT message can be used to actively establish a TCP connection to a remote partner. A typical example would be a TCP client application connecting to a remote server.

The command message takes the IP address and port number of the remote station as well as the handle of a previously opened socket as parameters. Additionally, timeout values for the connection establishment process itself and for later data send commands can be given.

	Command Message			
	Parameter	Type	Value	Description
Message Header	msg.rx	USIGN8	6	Identification of Receiver TCP_UDP task
	msg.tx	USIGN8	16	Identification of Transmitter User application
	msg.ln	USIGN8	16	Message Length Length in octets
	msg.nr	USIGN8	0 ... 255	Message Identification Unique number
	msg.a	USIGN8	0	Reply Identification No reply
	msg.f	USIGN8	0	Error Number No error
	msg.b	USIGN8	51	Command Identification TCP_CMD_CONNECT command
	msg.e	USIGN8	0	Extension Standard
Message Data	msg.d[0]	USIGN32		IP Address IP address of server to connect to
	msg.d[4]	USIGN16	0 ...65535	Port Number Port number of server to connect to
	msg.d[6]	USIGN8	0 ... 255	Socket Handle Handle of socket to connect to
	msg.d[7]	USIGN8	0	Reserved Reserved for future use
	msg.d[8]	USIGN32	see below	Send Timeout Timeout for future send commands
	msg.d[12]	USIGN32	see below	Connect Timeout Timeout for connect command

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

Send Timeout	Description
0	Default timeout (31000 milliseconds)
1 ... 2147483647	Wait up to specified time for successful transfer data (time in milliseconds)

Connect Timeout	Description
0	Default timeout (31000 milliseconds)
1 ... 2147483647	Wait up to specified time for connection (time in milliseconds)

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

The TCP_UDP task responds with the following message:

	Answer Message			
	Parameter	Type	Value	Description
Message Header	msg.rx	USIGN8	16	Identification of Receiver User application
	msg.tx	USIGN8	6	Identification of Transmitter TCP_UDP task
	msg.ln	USIGN8	8	Message Length Length in octets
	msg.nr	USIGN8	0 ... 255	Message Identification Unique number
	msg.a	USIGN8	51	Reply Identification TCP_CMD_CONNECT
	msg.f	USIGN8	0 110 111 112 113 115 118 119 121 151 152 158 200	Error Number No error Timeout expired Invalid timeout parameter Invalid socket handle Not possible in current socket state Destination unreachable Invalid IP address Invalid port Connection reset Invalid message length Unknown message command Command already running Task not initialized
	msg.b	USIGN8	0	Command Identification No command
	msg.e	USIGN8	0	Extension Standard
Message Data	msg.d[0]	USIGN32		IP Address IP address of remote client
	msg.d[4]	USIGN16	0...65535	Port Number Port number of remote client
	msg.d[6]	USIGN8	0 ... 255	Socket Handle Handle of socket for the incoming connection
	msg.d[7]	USIGN8	0	Reserved Reserved for future use

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

5.2.6 Sending TCP Data – TCP_CMD_SEND

The TCP_CMD_SEND command message can be used to transfer data to the TCP communication partner. The only required parameter besides the actual data to be transfered is a handle of a socket in established state. That means, this socket has to be connected to the remote communication parter using the TCP_CMD_WAIT_CONNECT or TCP_CMD_CONNECT commands beforehand.

Two message transfer modes can be used depending on the amount of data that need to be sent.

Up to 247 bytes of data can be sent in a single message. The **msg.e** parameter of the message should be set to zero as usually. The command message will be replied to with an answer message by the device when the data has been acknowledged by the remote TCP/IP stack. Afterwards the next TCP_CMD_SEND message can be sent to the device. Each data block sent this way is subject to timeout supervision. If the data cannot be delivered within the time specified in the TCP_CMD_WAIT_CONNECT or TCP_CMD_CONNECT command the connection will be aborted.

If the application needs to send data blocks larger than 247 bytes, the sequenced message transfer mode can be used. This mode comprises of three different types of messages. Each of these messages can transport up to 247 bytes of user data. In the device these data portions will be combined into larger blocks which will be sent through the network. A start message with **msg.e** set to 4 and an arbitrary value in **msg.nr** marks the beginning of a sequence of messages. Thereafter as many messages as required for the data transfer can be sent provided that the following conditions are met: Each of these messages must have **msg.e** set to 8 to mark it as an intermediate message. Additionally, the **msg.nr** field must be increased by one compared to the previous message. To end a sequenced message transfer a message with **msg.e** set to 12 must be sent. The handling of the **msg.nr** field is the same as for intermediate messages.

The device starts to send the accumulated data, as soon as a full size TCP segment can be assembled, or the last message of the sequence is recognized. Once the last message is processed the device waits for the remote TCP/IP stack to acknowledge the last data packet sent. A single TCP_CMD_SEND answer message is returned to the application then.

Some care should be taken when using the sequenced message transfer mechanism. There is a limitation of the number of messages that can be stored in the device. The device will not accept more messages from the application, if this limit is reached. On the other hand, the number of messages used in the device decreases, when it sends data through the network, or the application fetches messages, that are addressed to it. Thus, the application program should check for available messages after each single message that is sent during a sequenced message transfer. This is recommended anyway, because it is the only way to detect errors during the sequenced message transfer.

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

The format of a TCP_CMD_SEND command message is shown in the table below:

	Command Message			
	Parameter	Type	Value	Description
Message Header	msg.rx	USIGN8	6	Identification of Receiver TCP_UDP task
	msg.tx	USIGN8	16	Identification of Transmitter User application
	msg.ln	USIGN8	9 ... 255	Message Length Length in octets
	msg.nr	USIGN8	0 ... 255	Message Identification Unique number or msg.nr of the previous message increased by one in case of a sequenced message transfer
	msg.a	USIGN8	0	Reply Identification No reply
	msg.f	USIGN8	0	Error Number No error
	msg.b	USIGN8	52	Command Identification TCP_CMD_SEND command
	msg.e	USIGN8	0 4 8 12	Extension Single message First message of a sequence of messages Next message of a sequence of messages Last message of a sequence of messages
Message Data	msg.d[0]	USIGN32	0	Reserved Reserved for future use
	msg.d[4]	USIGN16	0	Reserved Reserved for future use
	msg.d[6]	USIGN8	0 ... 255	Socket handle Handle of socket
	msg.d[7]	USIGN8	see below	Options Options
	msg.d[8 ...]	USIGN8	0 ... 255	Data Data

Options of TCP_CMD_SEND command:

Options (Bit 0 ... 7)							
D7	D6	D5	D4	D3	D2	D1	D0
Reserved for future use							

The following table defines the layout of TCP_CMD_SEND answer message.

	Answer Message			
	Parameter	Type	Value	Description
Message Header	msg.rx	USIGN8	16	Identification of Receiver User application
	msg.tx	USIGN8	6	Identification of Transmitter TCP_UDP task
	msg.ln	USIGN8	8	Message Length Length in octets
	msg.nr	USIGN8	0 ... 255	Message Identification Unique number
	msg.a	USIGN8	52	Reply Identification TCP_CMD_SEND
	msg.f	USIGN8	0 110 112 113 116 120 121 151 152 156 158 200	Error Number No error Timeout expired Invalid socket handle Not possible in current socket state Invalid option parameter Connection closed Connection reset Invalid message length Unknown message command Message sequence error Command already running Task not initialized
	msg.b	USIGN8	0	Command Identification No command
	msg.e	USIGN8	0	Extension Standard
Message Data	msg.d[0]	USIGN32	0	Reserved Reserved for future use
	msg.d[4]	USIGN16	0	Reserved Reserved for future use
	msg.d[6]	USIGN8	0 ... 255	Socket handle Handle of socket
	msg.d[7]	USIGN8	0	Reserved Reserved for future use

NOTE: If the error code 110 (Timeout expired) is returned, the timeout for the data send operation expired, and the connection was aborted.

All other error codes indicate that just the current message was rejected. A possibly running sequenced message transfer would not be affected.

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

5.2.7 Sending UDP Data – UDP_CMD_SEND

The UDP_CMD_SEND command message can be used to transfer data to an UDP communication partner. The target's IP address and port number, and the handle of an UDP socket opened with the TCP_UDP_CMD_OPEN command must be provided.

Dependent on the amount of data to be sent, two types of message transfer are available. Up to 247 bytes of data can be sent in a single message. The **msg.e** parameter of the message should be set to zero as usually. The data from this message is sent in a UDP packet through the network, and an answer message is returned immediately.

In case data blocks larger than 247 bytes are to be sent, the sequenced message transfer mode can be used. This mode comprises of three different types of messages. Each of these messages can transport up to 247 bytes of user data. In the device these data portions will be assembled into one large UDP packet which is sent to the remote station. A start message with **msg.e** set to 4 and an arbitrary value in **msg.nr** marks the beginning of a sequence of messages. Thereafter up to four intermediate messages can be sent provided the following conditions are met: Each of these messages must have **msg.e** set to 8 to mark it as an intermediate message. Additionally, the **msg.nr** field must be increased by one compared to the previous message. To end the sequenced message transfer a message with **msg.e** set to 12 must be sent. The handling of the **msg.nr** field is the same as for intermediate messages.

When the last message of the sequence is recognized by the device, it sends the UDP packet and returns the answer message to the application afterwards.

NOTE: Please note the following limitations, when sending UDP data: Up to 1472 bytes can be sent per UDP packet. Thus the number of bytes sent during a sequenced messages transfer must stay within this limit, too. Additionally, the number of messages that can be queued before sending a UDP packet is limited to 6. If one of these two limits is exceeded, the sequenced message transfer is aborted, and an answer messsage holding an error code is returned. No data is sent on the network in this case.

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

Definition of a UDP_CMD_SEND command message:

	Command Message			
	Parameter	Type	Value	Description
Message Header	msg.rx	USIGN8	6	Identification of Receiver TCP_UDP task
	msg.tx	USIGN8	16	Identification of Transmitter User application
	msg.ln	USIGN8	9 ... 255	Message Length Length in octets
	msg.nr	USIGN8	0 ... 255	Message Identification Unique number or msg.nr of the previous message increased by one in case of a sequenced message transfer
	msg.a	USIGN8	0	Reply Identification No reply
	msg.f	USIGN8	0	Error Number No error
	msg.b	USIGN8	53	Command Identification UDP_CMD_SEND command
	msg.e	USIGN8	0 4 8 12	Extension Single message First message of a sequence of messages Next message of a sequence of messages Last message of a sequence of messages
Message Data	msg.d[0]	USIGN32		IP address Target IP address
	msg.d[4]	USIGN16	0 ...65535	Port Number Target Port number
	msg.d[6]	USIGN8	0 ... 255	Socket handle Handle of socket
	msg.d[7]	USIGN8	see below	Options Options
	msg.d[8 ...]	USIGN8	0 ... 255	Data Data

Options of UDP_CMD_SEND command:

Options (Bit 0 ... 7)							
D7	D6	D5	D4	D3	D2	D1	D0
Reserved for future use							

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

The TCP_UDP task responds with the following message:

	Answer Message			
	Parameter	Type	Value	Description
Message Header	msg.rx	USIGN8	16	Identification of Receiver User application
	msg.tx	USIGN8	6	Identification of Transmitter TCP_UDP task
	msg.ln	USIGN8	8	Message Length Length in octets
	msg.nr	USIGN8	0 ... 255	Message Identification Unique number
	msg.a	USIGN8	53	Reply Identification UDP_CMD_SEND
	msg.f	USIGN8	0 112 115 116 118 131 132 151 152 156 158 200	Error Number No error Invalid socket handle Destination is unreachable Invalid option parameter Invalid IP address Maximum data length exceeded Maximum number of queued message exceeded Invalid message length Unknown message command Message sequence error Command already running Task not initialized
	msg.b	USIGN8	0	Command Identification No command
	msg.e	USIGN8	0	Extension Standard
Message Data	msg.d[0]	USIGN32	0	Reserved Reserved for future use
	msg.d[4]	USIGN16	0	Reserved Reserved for future use
	msg.d[6]	USIGN8	0 ... 255	Socket handle Handle of socket
	msg.d[7]	USIGN8	0	Reserved Reserved for future use

NOTE: An answer message reporting no error does not guarantee that the data was successfully received by the communication partner.

The error code 115 (Destination is unreachable) is returned, if the previous UDP send command was targeted to the same destination as the current one, and a destination unreachable notification was received. So the **msg.f** field indicates an error from the previous send command in this case. However, the data from the current UDP send command have been transmitted on the line despite of this error.

If error code 131 (Maximum data length exceeded), or 132 (Maximum number of queued messages exceeded) is returned, the currently running sequenced message transfer has been aborted, and no data was sent.

In all other cases just the current message was rejected for the reason indicated. A possibly running sequenced message transfer would not be affected.

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

5.2.8 Setting Socket Options – TCP_UDP_CMD_SET_SOCK_OPTION

Socket specific parameters can be provided for each socket by sending a TCP_UDP_CMD_SET_SOCK_OPTION command message to the TCP_UDP task.

	Command Message			
	Parameter	Type	Value	Description
Message Header	msg.rx	USIGN8	6	Identification of Receiver TCP_UDP task
	msg.tx	USIGN8	16	Identification of Transmitter User application
	msg.ln	USIGN8	see below	Message Length Length in octets
	msg.nr	USIGN8	0 ... 255	Message Identification Unique number
	msg.a	USIGN8	0	Reply Identification No reply
	msg.f	USIGN8	0	Error Number No error
	msg.b	USIGN8	54	Command Identification TCP_UDP_CMD_SET_SOCK_OPTION
	msg.e	USIGN8	0	Extension Standard
Message Data	msg.d[0]	USIGN32	0	Reserved Reserved for future use
	msg.d[4]	USIGN16	0	Reserved Reserved for future use
	msg.d[6]	USIGN8	0 ... 255	Socket handle Handle of socket
	msg.d[7]	USIGN8	see below	Mode Type of option
	msg.d[8 ...]	see below	see below	Data Data required by option

Mode	Description	Data Type	Value	msg.ln
1	TCP_SOCK_TTL Set TTL value for outgoing IP packets	USIGN8	1 ... 255	9
2	TCP_SOCK_SEND_TIMEOUT Set new send timeout (milliseconds) **Applicable to TCP sockets only**	USIGN32	0 ... 2147483647 (0 – defaults to 31000)	12
5	TCP_SOCK_INACTIVE_TIMEOUT Set inactivity timeout (milliseconds) **Applicable to TCP sockets only**	USIGN32	0 ... 2147483647 (0 – Timeout inactive)	12
6	TCP_SOCK_KEEPALIVE_TIMEOUT Set keep-alive timeout (milliseconds) **Applicable to TCP sockets only**	USIGN32	0 ... 2147483647 (0 – Keep-alive inactive)	12
7	TCP_SOCK_ADD_MEMBERSHIP Add IP multicast group membership **Applicable to UDP sockets only**	USIGN32	Multicast group address	12
8	TCP_SOCK_DROP_MEMBERSHIP Drop IP multicast group membership	USIGN32	Multicast group address	12

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

	Applicable to UDP sockets only			
9	TCP_SOCK_MULTICAST_TTL Set TTL for multicast packets **Applicable to UDP sockets only**	USIGN8	1 ... 255	9
10	TCP_SOCK_MULTICAST_LOOP Set loopback mode of outgoing multicast packets **Applicable to UDP sockets only**	USIGN8	0 ... 1 (0 – Loopback inactive)	9

NOTE: The TCP_SOCK_INACTIVE_TIMEOUT and the TCP_SOCK_KEEPALIVE_-TIMEOUT option influence each other. Only one can be active at a time. Enabling one will automatically disable the other.

If either the inactivity timeout expires or the keep-alive mechanism fails, the connection will be closed automatically. The device will notify the application by sending a `TCP_UDP_CMD_RECEIVE` message containing an appropriate error code. Please refer to section 5.2.10 ‘Receiving TCP Data and UDP Data – `TCP_UDP_CMD_RECEIVE`’ for a description of the message.

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

The TCP_UDP task responds with the following message:

	Answer Message			
	Parameter	Type	Value	Description
Message Header	msg.rx	USIGN8	16	Identification of Receiver User application
	msg.tx	USIGN8	6	Identification of Transmitter TCP_UDP task
	msg.ln	USIGN8	8	Message Length Length in octets
	msg.nr	USIGN8	0 ... 255	Message Identification Unique number
	msg.a	USIGN8	54	Reply Identification TCP_UDP_CMD_SET_SOCK_OPTION
	msg.f	USIGN8	0 111 112 117 118 130 133 151 152 200	Error Number No error Invalid timeout parameter Invalid socket handle Invalid parameter Invalid IP address Invalid mode parameter Max. number of IP multicast groups exceeded Invalid message length Unknown message command Task not initialized
	msg.b	USIGN8	0	Command Identification No command
	msg.e	USIGN8	0	Extension Standard
Message Data	msg.d[0]	USIGN32	0	Reserved Reserved for future use
	msg.d[4]	USIGN16	0	Reserved Reserved for future use
	msg.d[6]	USIGN8	0 ... 255	Socket handle Handle of socket
	msg.d[7]	USIGN8	see below	Mode Type of option

Mode	Description
1	TCP_SOCK_TTL Set TTL value for outgoing IP packets
2	TCP_SOCK_SEND_TIMEOUT Set new send timeout
5	TCP_SOCK_INACTIVE_TIMEOUT Set inactivity timeout
6	TCP_SOCK_KEEPALIVE_TIMEOUT Set keep-alive timeout
7	TCP_SOCK_ADD_MEMBERSHIP Add IP multicast group membership
8	TCP_SOCK_DROP_MEMBERSHIP Drop IP multicast group membership
9	TCP_SOCK_MULTICAST_TTL Set TTL for multicast packets

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

10	TCP_SOCK_MULTICAST_LOOP Set loopback mode of outgoing multicast packets

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

5.2.9 Obtaining Socket Options – TCP_UDP_CMD_GET_SOCK_OPTION

Specific information on the individual socket can be obtained from the TCP_UDP task by sending a TCP_UDP_CMD_GET_SOCK_OPTION command message.

	Command Message			
	Parameter	Type	Value	Description
Message Header	msg.rx	USIGN8	6	Identification of Receiver TCP_UDP task
	msg.tx	USIGN8	16	Identification of Transmitter User application
	msg.ln	USIGN8	8	Message Length Length in octets
	msg.nr	USIGN8	0 ... 255	Message Identification Unique number
	msg.a	USIGN8	0	Reply Identification No reply
	msg.f	USIGN8	0	Error Number No error
	msg.b	USIGN8	55	Command Identification TCP_UDP_CMD_GET_SOCK_OPTION
	msg.e	USIGN8	0	Extension Standard
Message Data	msg.d[0]	USIGN32	0	Reserved Reserved for future use
	msg.d[4]	USIGN16	0	Reserved Reserved for future use
	msg.d[6]	USIGN8	0 ... 255	Socket handle Handle of socket
	msg.d[7]	USIGN8	see below	Mode Type of option

Mode	Description
1	TCP_SOCK_TTL Get TTL value of outgoing IP packets
2	TCP_SOCK_SEND_TIMEOUT Get current send timeout **Applicable to TCP sockets only**
3	TCP_SOCK_PROTOCOL Get protocol type
4	TCP_SOCK_PORT Get port number
5	TCP_SOCK_INACTIVE_TIMEOUT Get inactivity timeout **Applicable to TCP sockets only**
6	TCP_SOCK_KEEPALIVE_TIMEOUT Get keep-alive timeout **Applicable to TCP sockets only**
9	TCP_SOCK_MULTICAST_TTL Get TTL of multicast packets **Applicable to UDP sockets only**

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

10	TCP_SOCK_MULTICAST_LOOP Get loopback mode of outgoing multicast packets **Applicable to UDP sockets only**

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

The TCP_UDP task will return the requested information in an answer message as defined below.

	Answer Message			
	Parameter	Type	Value	Description
Message Header	msg.rx	USIGN8	16	Identification of Receiver User application
	msg.tx	USIGN8	6	Identification of Transmitter TCP_UDP task
	msg.ln	USIGN8	see below	Message Length Length in octets
	msg.nr	USIGN8	0 ... 255	Message Identification Unique number
	msg.a	USIGN8	55	Reply Identification TCP_UDP_CMD_GET_SOCK_OPTION
	msg.f	USIGN8	0 112 130 151 152 200	Error Number No error Invalid socket handle Invalid mode parameter Invalid message length Unknown message command Task not initialized
	msg.b	USIGN8	0	Command Identification No command
	msg.e	USIGN8	0	Extension Standard
Message Data	msg.d[0]	USIGN32	0	Reserved Reserved for future use
	msg.d[4]	USIGN16	0	Reserved Reserved for future use
	msg.d[6]	USIGN8	0 ... 255	Socket handle Handle of socket
	msg.d[7]	USIGN8	see below	Mode Type of option
	msg.d[8 ...]	see below	see below	Data Result data

Mode	Description	Data Type	Value	msg.ln
1	TCP_SOCK_TTL TTL value of outgoing IP packets	USIGN8	1 ... 255	9
2	TCP_SOCK_SEND_TIMEOUT Current send timeout (milliseconds)	USIGN32	1 ... 2147483647	12
3	TCP_SOCK_PROTOCOL Protocol type	USIGN8	1 (TCP_PROTO_TCP) 2 (TCP_PROTO_UDP)	9
4	TCP_SOCK_PORT Local port number	USIGN16	0 ... 65535	10
5	TCP_SOCK_INACTIVE_TIMEOUT Inactivity timeout (milliseconds)	USIGN32	0 ... 2147483647	12
6	TCP_SOCK_KEEPALIVE_TIMEOUT Keep-alive timeout (milliseconds)	USIGN32	0 ... 2147483647	12
9	TCP_SOCK_MULTICAST_TTL Get TTL value of multicast packets	USIGN8	1 ... 255	9
10	TCP_SOCK_MULTICAST_LOOP Get loopback mode of outgoing	USIGN8	0 ... 1	9

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

	multicast packets		(0 – Loopback inactive)	

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

5.2.10 Receiving TCP Data and UDP Data – TCP_UDP_CMD_RECEIVE

Data received from the network for a TCP socket or an UDP socket will be sent to the application in a command message according to the table below.

When receiving UDP packets which contain more than 247 bytes of data, the data from these packets is sent to the application in several consecutive messages. Such messages are marked by special values in bit 2 and bit 3 of the **msg.e** field. A value of 4 identifies the first message of the sequence. All subsequent messages contain the value 8 in the **msg.e** field. The value of 12 indicates the last message of the sequence. The **msg.nr** is incremented for each message of the sequence.

NOTE: Please note, that the sequenced message transfer does not exist for TCP receive data. All messages resulting from an incoming TCP data stream are sent to the application by means of individual messages. The value of **msg.nr** is incremented for each message.

Bit 1 of **msg.e** is set to one in every receive data message, to tell the application not to return an answer message.

An additional message with **msg.f** holding an error code will be sent to the application in case of a connection being closed or reset. This message can be treated as an end-of-data marker, because no more receive data messages from this socket will be sent to the application afterwards.

The table below defines the layout of a receive data message.

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

Definition of a receive data message:

	Command Message			
	Parameter	Type	Value	Description
Message Header	msg.rx	USIGN8	16	Identification of Receiver User application
	msg.tx	USIGN8	6	Identification of Transmitter TCP_UDP task
	msg.ln	USIGN8	8 ... 255	Message Length Length in octets
	msg.nr	USIGN8	0 ... 255	Message Identification Unique number
	msg.a	USIGN8	0	Reply Identification Not a reply
	msg.f	USIGN8	0 120 121	Error Number No error Connection closed Connection reset
	msg.b	USIGN8	17	Command Identification TCP_UDP_CMD_RECEIVE
	msg.e	USIGN8	0 + 2 4 + 2 8 + 2 12 + 2	Extension Single message, no answer message required First message, no answer message required Next message, no answer message required Last message, no answer message required
Message Data	msg.d[0]	USIGN32		IP address Originating IP address
	msg.d[4]	USIGN16	0 ...65535	Port Number Originating port Number
	msg.d[6]	USIGN8	0 ... 255	Socket handle Handle of socket
	msg.d[7]	USIGN8	see below	Option Options of received data
	msg.d[8 ...]	USIGN8	0 ... 255	Data Received data

Options of `TCP_UDP_CMD_RECEIVE` command:

Options (Bit 0 ... 7)							
D7	D6	D5	D4	D3	D2	D1	D0
Reserved for future use							

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

5.2.11 Shutdown of the Device – TCP_UDP_CMD_SHUTDOWN

In some situations the device is required to stop using the current IP address, and thus it has to stop communicating on the network. This can be caused by a DHCP lease expiring, or by a IP_CMD_SET_CONFIG message being received by the device.

In these cases the device will send a TCP_UDP_CMD_SHUTDOWN command message to the application. This message will tell the application to close all sockets, and to return a TCP_UDP_CMD_SHUTDOWN answer message to the device afterwards. Please look at the tables below for the definitions of both the command message and the answer message.

If the device doesn't receive the answer message within 30 seconds after sending the TCP_UDP_CMD_SHUTDOWN command, it will close all sockets automatically.

The device will then reenter initializing state, and all further command messages will be rejected. Please refer to the section 3 'Start-up of the Device' for a description of the device's initialization procedure.

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

TCP_UDP_CMD_SHUTDOWN command message:

	Command Message			
	Parameter	Type	Value	Description
Message Header	msg.rx	USIGN8	16	Identification of Receiver User application
	msg.tx	USIGN8	6	Identification of Transmitter TCP_UDP task
	msg.ln	USIGN8	0	Message Length Length in octets
	msg.nr	USIGN8	0 ... 255	Message Identification Unique number
	msg.a	USIGN8	0	Reply Identification No reply
	msg.f	USIGN8	0	Error Number No error
	msg.b	USIGN8	57	Command Identification TCP_UDP_CMD_SHUTDOWN
	msg.e	USIGN8	0	Extension Standard

TCP_UDP_CMD_SHUTDOWN answer message:

	Answer Message			
	Parameter	Type	Value	Description
Message Header	msg.rx	USIGN8	6	Identification of Receiver TCP_UDP task
	msg.tx	USIGN8	16	Identification of Transmitter User application
	msg.ln	USIGN8	0	Message Length Length in octets
	msg.nr	USIGN8	0 ... 255	Message Identification Unique number
	msg.a	USIGN8	57	Reply Identification TCP_UDP_CMD_SHUTDOWN
	msg.f	USIGN8	0	Error Number No error
	msg.b	USIGN8	0	Command Identification No command
	msg.e	USIGN8	0	Extension Standard

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

6 Error Codes

6.1 IP Task Error Codes

6.1.1 Initialization Error Codes

Error Number	Description
50	IP_ERR_INIT_NO_TCP_TASK TCP/UDP task not available
51	IP_ERR_INIT_TASK_CONFIG Internal task configuration data not available
52	IP_ERR_INIT_NO_ETHERNET_ADDR No Ethernet address (MAC address) available
53	IP_ERR_INIT_WAIT_FOR_WARMSTART Waiting for warm-start by application program
54	IP_ERR_INIT_INVALID_FLAG Unknown flag in start parameters
55	IP_ERR_INIT_INVALID_IP_ADDR Invalid IP address in start parameters
56	IP_ERR_INIT_INVALID_NETMASK Invalid subnet mask in start parameters
57	IP_ERR_INIT_INVALID_GATEWAY Invalid gateway IP address in start parameters
59	IP_ERR_INIT_UNKNOWN_HARDWARE Device type is unknown
60	IP_ERR_INIT_NO_IP_ADDR Failed to obtain an IP address from the the specified source(s)
61	IP_ERR_INIT_DRIVER_FAILED Driver layer failed to initialize
62	IP_ERR_INIT_NO_IP_ADDR_CFG No source for an IP address (BOOTP, DHCP, IP address parameter) specified
210	TASK_F_DATABASE No configuration database available
212	TASK_F_DATABASE_READ Error while reading configuration database
213	TASK_F_STRUCTURE Error while registering diagnostics structure

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

6.1.2 Run-time Error Codes

Error Number	Description
0	IP_OK No error
110	IP_ERR_TIMEOUT Timeout expired
111	TCP_ERR_TIMEOUT_INVALID Invalid timeout parameter
115	IP_ERR_DEST_UNREACHABLE Destination IP address not reachable
118	IP_ERR_IP_ADDR_INVALID IP address is invalid or doesn’t match
124	IP_ERR_ETH_ADDR_INVALID Invalid Ethernet address (MAC adress)
130	IP_ERR_MODE_UNKNOWN Invalid mode parameter
131	IP_ERR_ARP_CACHE_FULL ARP cache is full
134	IP_ERR_ARP_ENTRY_NOT_FOUND ARP entry not found in ARP cache
151	TASK_F_MESSAGESIZE Invalid message length
152	TASK_F_MESSAGECOMMAND Unknown message command
156	TASK_F_MESSAGESEQUENCE Sequence error during segmented message transfer
158	TASK_F_MESSAGECOMMANDRUNNING Command cannot be executed, because the previous command is still running
200	TASK_F_NOT_INITIALIZED Task is not initialized

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

6.2 TCP_UDP Task Error Codes

6.2.1 Initialization Error Codes

Error Number	Description
50	TCP_ERR_INIT_NO_IP_TASK IP task not ready
51	TCP_ERR_INIT_TASK_CONFIG Internal task configuration data not available
218	TASK_F_MEM_ALLOC Not enough internal memory available

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

6.2.2 Run-time Error Codes

Error Number	Description
0	TCP_OK No error
110	TCP_ERR_TIMEOUT Timeout expired
111	TCP_ERR_TIMEOUT_INVALID Invalid timeout parameter
112	TCP_ERR_SOCKET_INVALID Invalid socket handle
113	TCP_ERR_SOCKET_STATE Command cannot be executed, because the socket is in an inappropriate state
115	TCP_ERR_DEST_UNREACHABLE Destination (host, network, or port) is unreachable
116	TCP_ERR_OPTION_NOT_SUPPORTED Invalid option parameter
117	TCP_ERR_PARAMETER_INVALID Invalid parameter in command
118	TCP_ERR_IP_ADDR_INVALID IP address is invalid or doesn’t match
119	TCP_ERR_PORT_INVALID Port is invalid or not available
120	TCP_ERR_CONN_CLOSED Connection closed
121	TCP_ERR_CONN_RESET Connection reset
122	TCP_ERR_PROTOCOL_UNKNOWN Invalid protocol parameter
123	TCP_ERR_NO_SOCKETS No socket handles available
130	TCP_ERR_MODE_UNKNOWN Invalid mode in command
131	TCP_ERR_MAX_DATA_LEN_EXCEEDED Maximum data length exceeded
132	TCP_ERR_MAX_MSG_CNT_EXCEEDED Maximum number of queued messages exceeded
133	TCP_ERR_MAX_GROUP_EXCEEDED Maximum number of IP multicast groups exceeded
149	TCP_ERR_UNEXP_ANSWER Unexpected answer message received
151	TASK_F_MESSAGESIZE Invalid message length
152	TASK_F_MESSAGECOMMAND Unknown message command
156	TASK_F_MESSAGESEQUENCE Sequence error during segmented message transfer
158	TASK_F_MESSAGECOMMANDRUNNING Command cannot be executed, because the previous command is still running
200	TASK_F_NOT_INITIALIZED Task is not initialized

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E

7 Specifications

The data below apply to TCP_UDP task version V01.210 and IP task version V01.110.

7.1 Supported Protocols

- ARP - Address Resolution Protocol (RFC 826)
- IP - Internet Protocol (RFC 791)
- ICMP - Internet Control Message Protocol (RFC 792)
- IGMPv2 - Internet Group Management Protocol, Version 2 (RFC 2236)
- TCP - Transmission Control Protocol (RFC 793, RFC 896)
- UDP - User Datagram Protocol (RFC 768)
- BOOTP - Bootstrap Protocol (RFC 951, RFC 1542, RFC 2132)
- DHCP - Dynamic Host Configuration Protocol (RFC 2131, RFC 2132)
- Ethernet frame types: Ethernet II (RFC 894), IEEE 802.3 receive only (RFC 1042)

7.2 Technical Data

- Number of sockets: 16
- ARP cache size: 64 entries
- ARP timeout: 600 seconds
- Route cache size: 32 entries
- Route timeout: 900 seconds
- IP multicast groups: receive: 64, send: not limited
- IP datagram size: up to 1500 bytes

7.3 Limitations

- IP fragmentation not supported
- TCP urgent data not supported
- TCP port 0 not supported
- UDP port 67 reserved for BOOTP and DHCP
- UDP port 25383 reserved for Hilscher NetIdent Protocol

Copyright * Hilscher Gesellschaft für Systemautomation * Pi:EN#5E