Protocol API

Socket Interface

Packet Interface

Hilscher Gesellschaft für Systemautomation mbH

www.hilscher.com

DOC140401API07EN | Revision 7 | English | 2021-04 | Released | Public

Table of contents

1 Introduction

1.1 About this document

This manual describes the application programming interface of the Socket Interface.

1.2 List of revisions

Rev	Date	Name	Chapter	Revision
4	2018-01-24	BM AM AT	1.5 3.2.4.1 3.2.12, 3.2.13 3.3	Remark added that lwIP-based loadable firmware is required. Wrong service introduction corrected. New Socket Options added. Section *Options structure definition* added.
5	2019-03-01	HHE	3.2	Section *Description of packets*: Structure tLL added in Table 4.
			3.2.14.2	Section *SOCK_GETIFADDRS_CNF packet* expanded.
			4	Section *Error codes and status codes* updated.
6	2020-02-11	ABE, HHE	1.7.2	Section *Limitations* updated.
			2.2.1	Section *Server application*: Sequence diagram added.
7	2020-04-07	BME	3.2.12 3.2.13 3.3	API changes and API extensions for *Set Socket Options service* and *Get Socket Options service*. Add description for each socket option.
		BME	1.5 1.7.2	Add netX100 as supported chip to *System requirements*. Add hint regarding firmware tag list to *Limitations*.
		ABE	3.2.11	Extend description of *Socket Control service* regarding new non-blocking mode.
		BME	2.2.3 3.2.10	Add information about IP address services for application and extend *Socket Poll service* accordingly.

Table 1: List of revisions

1.3 Terms, abbreviations and definitions

Term	Description
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
IP	Internet Protocol

Table 2: Terms, abbreviations and definitions

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

All IP addresses in this document have host byte order.

1.4 Functional overview

The socket interface provides application access to the IP implementation of the protocol firmware. It enables the application program to use TCP and UDP services over the network. The packet API of the socket interface was designed with standard BSD/POSIX socket function interface in mind. Some redundant interfaces are not implemented as they can be easily mapped onto implemented interfaces.

The following features are available

- Datagram based communication (UDP)
- Stream based communication (TCP)
- Standard poll functionality to wait for events on sockets

1.5 System requirements

This software package has the following environmental system requirements:

- Real-Time-Ethernet protocol loadable firmware with socket interface. The Packet API will be available through protocol stack DPM channel mailbox. The socket interface is only available for loadable firmware based on lwIP IP stack. If this is the case it is documented in protocol API Manual.
- netX51 / netX52 / netX90 / netX100 communication processor

1.6 Intended audience

This manual is suitable for software developers with the following background:

- Knowledge of the programming language C
- Knowledge of standard IP socket interface

1.7 Specifications

1.7.1 Technical data

- Capabilities and limitations
  - Transmit/receive buffer size per socket. This depends on the particular firmware and hardware. Please the consult firmware manual about details.
  - Number of active sockets depends on particular firmware and hardware. Please consult firmware manual about details.
  - UDP protocol message size up to 1472 bytes
  - TCP stream communication with support of passive sockets (listen sockets for server applications)
  - IPv4 protocol
  - Blocking and non-blocking socket mode
- Configuration
  - General IP configuration is done by Real-time Ethernet protocol stack. See corresponding protocol manual for details.
  - Socket specific options available
- Diagnostic
  - No explicit diagnostic functionality as there is no separate DPM channel.

1.7.2 Limitations

Most limits of the socket interface packet API depend on the particular protocol firmware and the used hardware. Details of this might be found in the corresponding protocol API manual. In particular the following properties are affected by limits

- Number of concurrent active sockets (may be changeable via tag list of particular firmware)
- **Size of socket receive and transmit queues (may be changeable via tag list of particular firmware)**
- Transmission speed

Furthermore, the following functionality is not available and not supported:

- IPv6 protocol is not supported

2 Socket communication

The Socket Concept provides a common method for communication between applications. Sockets can typically be used for local communication and for communication across networks. Sockets support different communication protocols and methods. The following sections will provide a short introduction into the protocols and methods supported by the Socket API.

2.1 User Datagram Protocol

The User Datagram Protocol (UDP) provides message-based communication on top of the IP protocol. It is a stateless protocol and thus cannot guarantee message delivery or order. Besides that, it has a low latency and supports addressing multiple destinations in a single message (broadcast). The handling of a UDP socket is almost similar for a client or a server application. Typically, the following steps are necessary to communicate via the UDP protocol:

1. Create a UDP Socket
2. Optionally bind the UDP Socket to a specific local port number. This step is required for server applications in order to make the server accessible under this port number. It will be performed automatically on the first send or receive request.
3. Use the socket for communication
   A. Wait for incoming data using receive or poll requests. Retrieve incoming data using receive requests
   B. Transmit outgoing data using send requests.
4. Close the socket

2.2 Transmission Control Protocol

The Transmission Control Protocol (TCP) provides stream-based communication on top of the IP protocol. It is connection-oriented and ensures the order of the exchanged data and the transmission. Consequently, it has a larger latency than UDP and does not support addressing multiple destinations. TCP requires different handling for connection establishment on server and client side.

2.2.1 Server application

For a TCP server application, perform the following steps:

1. Create a TCP Socket
2. Bind the socket to a specific local port number
3. Switch the socket into passive mode using a listen request. In that mode, the socket can handle incoming connections.
4. Use the socket to handle incoming connections: Wait for new incoming connections using accept or poll requests. Retrieve incoming connections using the accept request. Each new connection will be assigned to its own, new socket. Data exchange with a new connection is performed with this new socket as described below.
5. Close the socket

The following sequence diagram shows the packet flow:

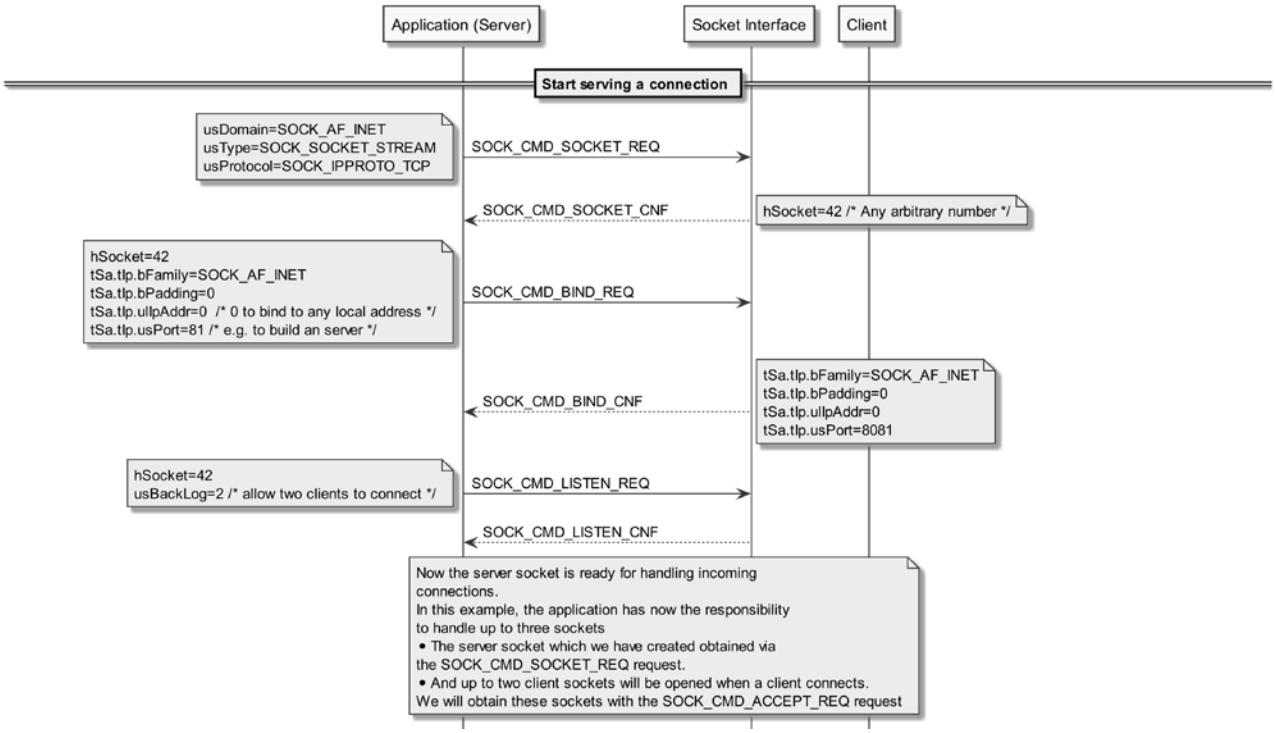

Figure 1: Server application sequence diagram (part 1)

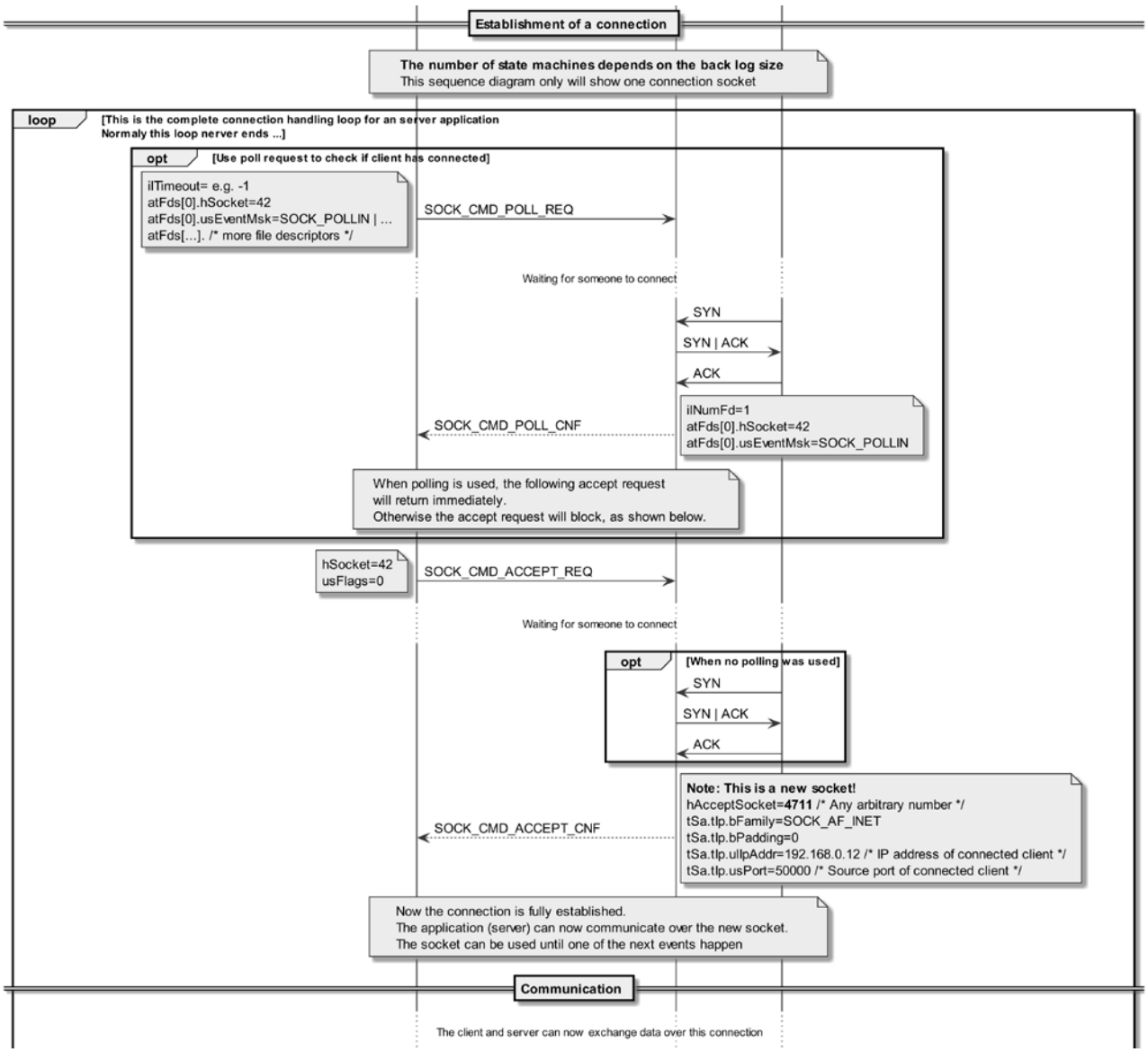

Figure 2: Server application sequence diagram (part 2)

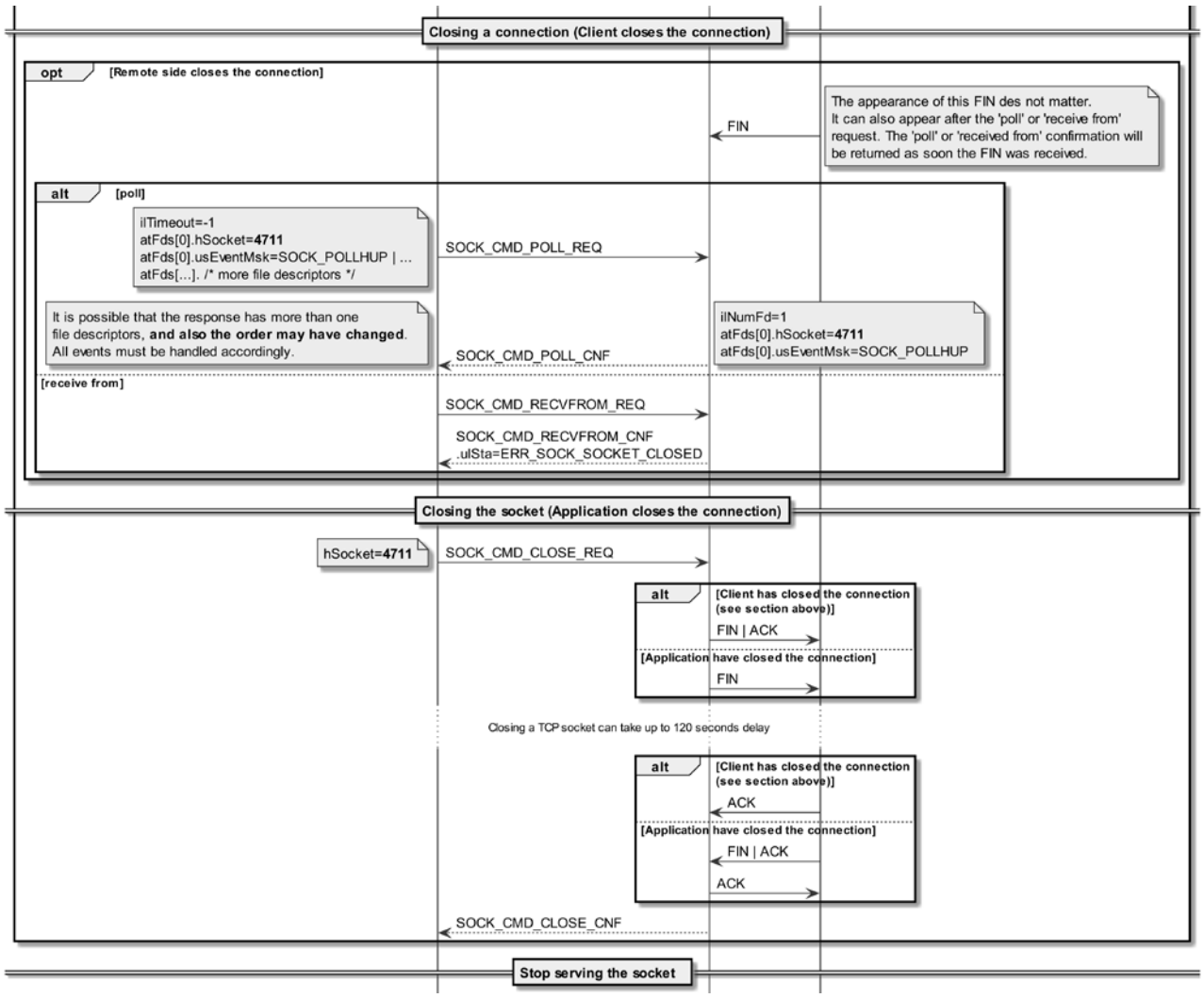

Figure 3: Server application sequence diagram (part 3)

2.2.2 Client application

In contrast to that, a TCP client application needs to perform the following steps to establish a connection:

1. Create a TCP Socket
2. Connect the socket to a remote IP address and port number

After connection establishment phase, the communication can take place:

1. Use the socket for communication
   A. Wait for incoming data using receive or poll requests. Retrieve incoming data using receive requests
   B. Transmit outgoing data using send requests.
2. Close the socket. Closing a connected socket will be signaled and synchronized with the remote application. Thus closing a TCP socket might not happen immediately.

2.2.3 IP Address change handling (application view)

An application may

- like to know the current IP address used by the IP stack or
- like to be informed when the IP address changes.

To fulfil these requirements, the stack provides services that the application can use.

Using the *Get Interface Addresses service*, the application can actively read the currently used IP address from the IP stack at any time.

Using the *Socket Poll service*, the application will be aware about changes of the IP address. To accomplish this, the Poll service offers a specific socket handle: `SOCK_HANDLE_IPV4CHANGE 0xF000`. After the application receives the information about the change of the IP address, it can read the current IP address using the *Get Interface Addresses service*.

Note, that there is no Socket API service available to the application to modify the IP address used by the IP stack. The IP address is set and controlled by the Real-time Ethernet protocol stack combined with the IP stack.

3 The application interface

This chapter describes the packets of the socket interface.

The firmware will forward any socket interface related messages to the socket API component. The socket API component will map the message to the corresponding functionality. The socket API component will not send any indications to the application. Thus registering an application is not required for the socket API itself. However, the protocol API might still require a registered application for its operation.

The firmware coordinates the routing of the socket API messages to the socket API component. The socket API component manages the active socket objects and handles the messages.

3.1 Overview

The socket interface offers the following services:

Topic	Section	Page
Socket management	Socket services	14
	Bind service	17
	Connect service	19
	Listen Service	21
	Accept service	23
	Socket Abort service	31
	Socket Control service	36
	Socket Close service	29
	Set Socket Options service	40
	Get Socket Options service	43
Data transfer	Receive From service	25
	Send To service	27
Polling	Socket Poll service	33
Identification	Get Interface Addresses service	46

Table 3: Overview of services

3.2 Description of packets

All packets of the socket API use the same default Hilscher packet header. The structure is defined as follows:

```
typedef struct SOCK_PCKHEADER_Ttag SOCK_PCKHEADER_T;

__PACKED_PRE struct __PACKED_POST SOCK_PCKHEADER_Ttag
{
  uint32_t  ulDest;   /* destination of the packet (task message queue reference) */
  uint32_t  ulSrc;    /* source of the packet (task message queue reference) */
  uint32_t  ulDestId; /* destination reference (internal use for message routing) */
  uint32_t  ulSrcId;  /* source reference (internal use for message routing) */
  uint32_t  ulLen;    /* length of packet data (starting from the end of the header) */
  uint32_t  ulId;     /* identification reference (internal use by the sender) */
  uint32_t  ulSta;    /* operation status code (error code, initialize with 0) */
  uint32_t  ulCmd;    /* operation command code */
  uint32_t  ulExt;    /* extension count (nonzero in multi-packet transfers) */
  uint32_t  ulRout;   /* router reference (internal use for message routing) */
};
```

The socket API uses a file descriptor similar to BSD/POSIX to refer to a socket. The type of the file descriptor is defined as follows:

```
typedef uint16_t SOCK_H;
```

Furthermore, communication endpoints are associated with an address. The address format depends on the socket domain. The current implementation only supports the layer 2 (hardware address) (identified by SOCK_AF_PACKET) and IPv4 internet domain (identified by SOCK_AF_INET). The following structures are defined for the address information:

```
typedef enum SOCK_SOCKET_DOMAIN_Etag SOCK_SOCKET_DOMAIN_E;

enum SOCK_SOCKET_DOMAIN_Etag
{
  SOCK_AF_INET = 2,
  SOCK_AF_PACKET = 17,
};

typedef struct SOCK_ADDR_COMMON_Ttag SOCK_ADDR_COMMON_T;

__PACKED_PRE struct __PACKED_POST SOCK_ADDR_COMMON_Ttag
{
  uint8_t    bFamily;

  uint8_t    bReserved[14];

  uint8_t    bPadding;
};

typedef struct SOCK_ADDR_IP_Ttag SOCK_ADDR_IP_T;

__PACKED_PRE struct __PACKED_POST SOCK_ADDR_IP_Ttag
{
  uint8_t    bFamily;

  uint8_t    bPadding;

  uint32_t   ulIpAddr;

  uint16_t   usPort;
};
```

```
typedef struct SOCK_ADDR_LL_Ttag SOCK_ADDR_LL_T;

__HIL_PACKED_PRE struct __HIL_PACKED_POST SOCK_ADDR_LL_Ttag
{
  /** Set to SOCK_AF_PACKET */
  uint8_t bFamily;
  /** index of the interface, starts from 1 (Zero means all interfaces) */
  uint8_t bIfIndex;
  /** Padding */
  uint8_t abPadding[4];
  /** reserved for future usage */
  uint8_t bPktType;
  /** Length of hardware address */
  uint8_t bLlAddrLen;
  /** Up to 8 bytes hardware address */
  uint8_t abLlAddr[8];
};

typedef union SOCK_ADDR_Ttag SOCK_ADDR_T;

union SOCK_ADDR_Ttag
{
  /** contains common fields of all socket address structures */
  SOCK_ADDR_COMMON_T tCommon;
  /** IPv4 specific socket address */
  SOCK_ADDR_IP_T     tIp;
  /** Packet (Layer 2) socket address */
  SOCK_ADDR_LL_T     tLL;
};
```

Area	Variable	Type	Value / Range	Description
tCommon	structure SOCK_ADDR_COMMON_T			
	bFamily	uint8_t		Socket Family / Domain
	bReserved	uint8_t[14]		Placeholder for family specific data
	bPadding	uint8_t		Padding for alignment
tIp	structure SOCK_ADDR_IP_T			
	bFamily	uint8_t	0x2	Socket Family SOCK_AF_INET
	bPadding	uint8_t	0	Padding for alignment of following fields
	ulIpAddr	uint32_t		IP Address of endpoint
	usPort	uint16_t		Port of endpoint
tLL	structure SOCK_ADDR_LL_T			
	bFamily	uint8_t	0x11	Socket Family SOCK_AF_PACKET
	bIfIndex	uint8_t		Reserved for future usage
	bPadding	uint8_t[4]	0	Padding for alignment of following fields
	bPktType	uint8_t		Reserved for future usage
	bLlAddrLen	uint8_t	1 .. 8	Length of the hardware address
	abLlAddr	uint8_t[8]		Up to 8 bytes hardware address

Table 4: Socket Address Union – SOCK_ADDR_T

3.2.1 Socket services

Communication requires a socket for identification. The application has to create a new socket for communication.

3.2.1.1 SOCK_CMD_SOCKET_REQ packet

The application shall send this packet to create a new socket. For valid parameter combinations, see Table 8 on page 15.

Packet structure reference

```
typedef enum SOCK_SOCKET_TYPE_Etag SOCK_SOCKET_TYPE_E;

enum SOCK_SOCKET_TYPE_Etag
{
  SOCK_SOCKET_STREAM = 1,
  SOCK_SOCKET_DGRAM  = 2,
};

typedef enum SOCK_IPPROTO_Etag SOCK_IPPROTO_E;

enum SOCK_IPPROTO_Etag
{
  /** Dummy, alias for TCP */
  SOCK_IPPROTO_IP  = 0,
  /** TCP */
  SOCK_IPPROTO_TCP = 6,
  /** UDP */
  SOCK_IPPROTO_UDP = 17,
};

typedef struct SOCK_SOCKET_REQ_DATA_Ttag SOCK_SOCKET_REQ_DATA_T;

__PACKED_PRE struct __PACKED_POST SOCK_SOCKET_REQ_DATA_Ttag
{
  /** the socket domain */
  uint32_t usDomain;
  /** the socket type */
  uint32_t usType;
  /** the socket protocol */
  uint32_t usProtocol;
};

typedef struct SOCK_SOCKET_REQ_Ttag SOCK_SOCKET_REQ_T;

__PACKED_PRE struct __PACKED_POST SOCK_SOCKET_REQ_Ttag
{
  SOCK_PCKHEADER_T       tHead;

  SOCK_SOCKET_REQ_DATA_T tData;
};
```

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination Queue-Handle, unchanged
ulLen	uint32_t	0x0000000C	Packet Data Length in bytes.
ulCmd	uint32_t	0x00009610	SOCK_CMD_SOCKET_REQ
Data			
ulDomain	uint32_t	0x00000002	Domain of the Socket. Currently only Internet Domain supported.
ulType	uint32_t	0x1 or 0x2	Socket Type. See Table 6
ulProtocol	uint32_t	0, 6 or 17	Socket Protocol Type. See Table 7

Table 5: SOCK_CMD_SOCKET_REQ – Packet

Socket Type Symbol	Value	Description
SOCK_SOCKET_STREAM	1	Stream connection
SOCK_SOCKET_DGRAM	2	Datagram connection

Table 6: Socket types

Protocol Symbol	Value	Description
SOCK_IPPROTO_IP	0	Alias for SOCK_IPPROTO_TCP
SOCK_IPPROTO_TCP	6	TCP Protocol
SOCK_IPPROTO_UDP	17	UDP Protocol

Table 7: Socket protocols

Domain	Socket Type	Protocol	Connection
SOCK_AF_INET	SOCK_SOCKET_STREAM	SOCK_IPPROTO_IP	TCP IPv4
SOCK_AF_INET	SOCK_SOCKET_STREAM	SOCK_IPPROTO_TCP	TCP IPv4
SOCK_AF_INET	SOCK_SOCKET_DGRAM	SOCK_IPPROTO_UDP	UDP IPv4

Table 8: Valid domain, Socket type and protocol combinations

3.2.1.2 SOCK_CMD_SOCKET_CNF packet

The stack will return this packet to the application as a confirmation to the SOCK_CMD_SOCKET_REQ packet. Depending on the status of the operation, the confirmation packet contains the new socket handle or an error code.

Packet structure reference

```
typedef struct SOCK_SOCKET_CNF_DATA_Ttag SOCK_SOCKET_CNF_DATA_T;

__PACKED_PRE struct __PACKED_POST SOCK_SOCKET_CNF_DATA_Ttag
{
  SOCK_H hSocket;
};

typedef struct SOCK_SOCKET_CNF_Ttag SOCK_SOCKET_CNF_T;

__PACKED_PRE struct __PACKED_POST_SOCK_SOCKET_CNF_Ttag
{
  SOCK_PCKHEADER_T       tHead;

  SOCK_SOCKET_CNF_DATA_T tData;
};
```

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination Queue-Handle, unchanged
ulLen	uint32_t	0x00000002	Packet Data Length in bytes
ulSta	uint32_t	0	See section *Error codes and status codes* on page 51.
ulCmd	uint32_t	0x00009611	SOCK_CMD_SOCKET_CNF
Data			
hSocket	uint16_t		Handle of created socket on success

Table 9: SOCK_CMD_SOCKET_CNF – Packet

3.2.2 Bind service

The application has to use this service to bind a socket to a local endpoint address. This service is mandatory for any server application. In case of Internet Domain, the IP address might be set to zero which means ‘any local IP address’. If the port number is set to zero, a port will be selected automatically. The assigned port number can be determined from the confirmation packet.

3.2.2.1 SOCK_CMD_BIND_REQ packet

The application shall send this packet in order to bind the socket to a local endpoint.

Packet structure reference

```
typedef struct SOCK_ADDR_DATA_Ttag SOCK_ADDR_DATA_T;

__PACKED_PRE struct __PACKED_POST SOCK_ADDR_DATA_Ttag
{
  SOCK_H   hSocket;

  SOCK_ADDR_T tSa;
};

typedef struct SOCK_BIND_REQ_Ttag SOCK_BIND_REQ_T;

__PACKED_PRE struct __PACKED_POST SOCK_BIND_REQ_Ttag
{
  SOCK_PCKHEADER_T     tHead;

  SOCK_ADDR_DATA_T tData;
};
```

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination Queue-Handle.
ulLen	uint32_t	0x00000012	Packet Data Length in bytes.
ulCmd	uint32_t	0x00009614	SOCK_CMD_BIND_REQ
Data			
hSocket	uint16_t		Handle of the socket to connect
tSa	Union		Local Endpoint Address to bind the socket to. See Table 4.

Table 10: SOCK_CMD_BIND_REQ – Packet

3.2.2.2 SOCK_CMD_BIND_CNF packet

The stack will return this packet to the application as a confirmation to the SOCK_CMD_BIND_REQ packet. If the IP Port was set to Zero in the request packet, the confirmation will contain the automatically assigned port number value.

Packet structure reference

```
typedef struct SOCK_BIND_CNF_Ttag SOCK_BIND_CNF_T;

__PACKED_PRE struct __PACKED_POST SOCK_BIND_CNF_Ttag
{
  SOCK_PCKHEADER_T     tHead;

  SOCK_ADDR_DATA_T tData;
};
;
```

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination Queue-Handle, unchanged
ulLen	uint32_t	0x00000012	Packet Data Length in bytes
ulSta	uint32_t		See section *Error codes and status codes* on page 51.
ulCmd	uint32_t	0x00009615	SOCK_CMD_BIND_CNF
Data			
hSocket	uint16_t		Handle of the socket to connect
tSa	Union		Local Endpoint Address to bind the socket to. See Table 4.

Table 11: SOCK_CMD_BIND_CNF – Packet

3.2.3 Connect service

The application has to use this service to connect a socket to a remote endpoint. This service is mandatory for TCP client applications.

3.2.3.1 SOCK_CMD_CONNECT_REQ packet

The application shall send this packet in order to connect a socket.

Packet structure reference

```
typedef struct SOCK_CONNECT_REQ_DATA_Ttag SOCK_CONNECT_REQ_DATA_T;

__PACKED_PRE struct __PACKED_POST SOCK_CONNECT_REQ_DATA_Ttag
{
  SOCK_H   hSocket;

  SOCK_ADDR_T tSa;
};

typedef struct SOCK_CONNECT_REQ_Ttag SOCK_CONNECT_REQ_T;

__PACKED_PRE struct __PACKED_POST SOCK_CONNECT_REQ_Ttag
{
  SOCK_PCKHEADER_T        tHead;

  SOCK_CONNECT_REQ_DATA_T tData;
};
```

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination Queue-Handle.
ulLen	uint32_t	0x00000012	Packet Data Length in bytes.
ulCmd	uint32_t	0x00009612	SOCK_CMD_CONNECT_REQ
Data			
hSocket	uint16_t		Handle of the socket to connect
tSa	Union		Remote Endpoint Address to connect the socket to. See Table 4.

Table 12: SOCK_CMD_CONNECT_REQ – Packet

3.2.3.2 SOCK_CMD_CONNECT_CNF packet

The stack will return this packet to the application as a confirmation to the SOCK_CMD_CONNECT_REQ packet.

Packet structure reference

```
typedef struct SOCK_EMPTY_PCK_Ttag SOCK_CONNECT_CNF_T;
```

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination Queue-Handle, unchanged
ulLen	uint32_t	0x00000000	Packet Data Length in bytes
ulSta	uint32_t		See section *Error codes and status codes* on page 51.
ulCmd	uint32_t	0x00009613	SOCK_CMD_CONNECT_CNF

Table 13: SOCK_CMD_CONNECT_CNF – Packet

3.2.4 Listen Service

The application can use the listen service to switch a socket into passive mode. This mode is relevant for TCP sockets in order to listen for incoming connections. Thus, this service is mandatory for TCP servers. This service is not applicable for UDP sockets.

3.2.4.1 SOCK_CMD_LISTEN_REQ packet

The application shall send this packet to switch the socket into passive mode. In passive mode, the socket will be waiting for incoming connections and create a new socket for each incoming connection. The resources for such a connection socket are taken from the listening socket's backlog (resource pool). This service is useful for connection-oriented sockets only.

Packet structure reference

```
typedef struct SOCK_LISTEN_REQ_DATA_Ttag SOCK_LISTEN_REQ_DATA_T;

struct SOCK_LISTEN_REQ_DATA_Ttag
{
  SOCK_H hSocket;
  /** maximum size of pending connect queue */
  uint16_t  usBackLog;
};

typedef struct SOCK_LISTEN_REQ_Ttag SOCK_LISTEN_REQ_T;

struct SOCK_LISTEN_REQ_Ttag
{
  SOCK_PCKHEADER_T       tHead;

  SOCK_LISTEN_REQ_DATA_T tData;
};
```

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination Queue-Handle.
ulLen	uint32_t	0x00000004	Packet Data Length in bytes.
ulCmd	uint32_t	0x00009616	SOCK_CMD_LISTEN_REQ
Data			
hSocket	uint16_t		Handle of the socket to switch into listen mode
usBackLog	uint16_t		Maximum number of pending and active connections to this socket. Attention: For any possible connection, a full socket will be pre-allocated within Socket API component. This amount of resources is part of the maximum socket count limit.

Table 14: SOCK_CMD_LISTEN_REQ – Packet

3.2.4.2 SOCK_CMD_LISTEN_CNF packet

The stack will return this packet to the application after reception of the SOCK_CMD_LISTEN_REQ packet.

Packet structure reference

```
typedef struct SOCK_EMPTY_PCK_Ttag SOCK_LISTEN_CNF_T;
```

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination Queue-Handle, unchanged
ulLen	uint32_t	0x00000000	Packet Data Length in bytes
ulSta	uint32_t		See section *Error codes and status codes* on page 51.
ulCmd	uint32_t	0x00009617	SOCK_CMD_LISTEN_CNF

Table 15: SOCK_CMD_LISTEN_CNF – Packet

3.2.5 Accept service

The application can use the accept service to accept a pending connection request on a connection oriented socket in listen mode (e. g TCP). This service is mandatory for TCP servers and not applicable for UDP sockets.

3.2.5.1 SOCK_CMD_ACCEPT_REQ packet

The application shall send this packet to the stack in order to accept a pending connection of a socket

Packet structure reference

```
typedef struct SOCK_ACCEPT_REQ_DATA_Ttag SOCK_ACCEPT_REQ_DATA_T;

__PACKED_PRE struct __PACKED_POST SOCK_ACCEPT_REQ_DATA_Ttag
{
  SOCK_H hSocket;

  uint16_t usFlags;
};

typedef struct SOCK_ACCEPT_REQ_Ttag SOCK_ACCEPT_REQ_T;

__PACKED_PRE struct __PACKED_POST SOCK_ACCEPT_REQ_Ttag
{
  SOCK_EMPTY_PCK_T       tHead;

  SOCK_ACCEPT_REQ_DATA_T tData;
};
```

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination Queue-Handle.
ulLen	uint32_t	0x00000004	Packet Data Length in bytes.
ulCmd	uint32_t	0x00009618	SOCK_CMD_ACCEPT_REQ
Data			
hSocket	uint16_t		Handle of a socket in listen mode
usFlags	uint16_t	0	Reserved for Future Usage Set to Zero.

Table 16: SOCK_CMD_ACCEPT_REQ – Packet

3.2.5.2 SOCK_CMD_ACCEPT_CNF packet

The stack will return this packet to the application on reception of the SOCK_CMD_ACCEPT_REQ packet when a new connection is available. If the socket is in Blocking Mode (default), the stack will not return this packet until the socket is closed or the stack receives a connection request.

Packet structure reference

```
typedef struct SOCK_ACCEPT_CNF_DATA_Ttag SOCK_ACCEPT_CNF_DATA_T;

__PACKED_PRE struct __PACKED_POST SOCK_ACCEPT_CNF_DATA_Ttag
{
  /** the socket handle of the new connection */
  SOCK_H   hAcceptSocket;
  /** address information */
  SOCK_ADDR_T tSa;
};

typedef struct SOCK_ACCEPT_CNF_Ttag SOCK_ACCEPT_CNF_T;

__PACKED_PRE struct __PACKED_POST SOCK_ACCEPT_CNF_Ttag
{
  SOCK_EMPTY_PCK_T       tHead;

  SOCK_ACCEPT_CNF_DATA_T tData;
};
```

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination Queue-Handle, unchanged
ulLen	uint32_t	0x00000012	Packet Data Length in bytes
ulSta	uint32_t		See section *Error codes and status codes* on page 51.
ulCmd	uint32_t	0x00009619	SOCK_CMD_ACCEPT_CNF
Data			
hSocket	uint16_t		Handle of the new socket associated with the new connection
tSa	Union		Remote Endpoint Address of the new connection. See Table 4.

Table 17: SOCK_CMD_ACCEPT_CNF – Packet

3.2.6 Receive From service

The application can use the Receive From Service to retrieve received data from a socket. By default, this service is blocking and waiting for receipt of data. You can change this behavior using the *Socket Control service*.

3.2.6.1 SOCK_CMD_RECVFROM_REQ packet

The application shall send this packet to the stack in order to retrieve received data from a socket.

Packet structure reference

```
typedef struct SOCK_RECVFROM_REQ_DATA_Ttag SOCK_RECVFROM_REQ_DATA_T;

__PACKED_PRE struct __PACKED_POST SOCK_RECVFROM_REQ_DATA_Ttag
{
  SOCK_H hSocket;
  /** Reserved for future extensions set to zero */
  uint16_t  usFlags;

  uint16_t  usMaxLen;
};

typedef struct SOCK_RECVFROM_REQ_Ttag SOCK_RECVFROM_REQ_T;
__PACKED_PRE struct __PACKED_POST SOCK_RECVFROM_REQ_Ttag
{
  SOCK_PCKHEADER_T         tHead;

  SOCK_RECVFROM_REQ_DATA_T tData;
};
```

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination Queue-Handle.
ulLen	uint32_t	0x00000006	Packet Data Length in bytes.
ulCmd	uint32_t	0x0000961A	SOCK_CMD_RECVFROM_REQ
Data			
hSocket	uint16_t		Handle of a socket to retrieve data from
usFlags	uint16_t	0	Receive Flags. Reserved for future. Set to zero.
usMaxLen	uint16_t	1-1472	Maximum Number of Receive Data to return in Confirmation. If this length is smaller than a pending datagram for Datagram Sockets the excess data will be dropped.

Table 18: SOCK_CMD_RECVFROM_REQ – Packet

3.2.6.2 SOCK_CMD_RECVFROM_CNF packet

The stack will return this packet to the application after reception of the SOCK_CMD_RECVFROM_REQ packet when data is available for the socket. If the socket is in Blocking Mode (default), the stack will not return this packet until the socket is closed or data is available.

Packet structure reference

```
typedef struct SOCK_RECVFROM_CNF_DATA_Ttag SOCK_RECVFROM_CNF_DATA_T;

__PACKED_PRE struct __PACKED_POST SOCK_RECVFROM_CNF_DATA_Ttag
{
  /** remote address information */
  SOCK_ADDR_T  tSa;
  /** data */
  uint8_t      abBuffer[SOCK_LIMITS_MAX_PCK_SIZE];
};

typedef struct SOCK_RECVFROM_CNF_Ttag SOCK_RECVFROM_CNF_T;

__PACKED_PRE struct __PACKED_POST SOCK_RECVFROM_CNF_Ttag
{
  SOCK_PCKHEADER_T         tHead;

  SOCK_RECVFROM_CNF_DATA_T tData;
};
```

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination Queue-Handle, unchanged
ulLen	uint32_t	0x000000010 - …	Packet Data Length in bytes. Depends on the amount of received data
ulSta	uint32_t		See section *Error codes and status codes* on page 51.
ulCmd	uint32_t	0x0000961B	SOCK_CMD_RECVFROM_CNF
Data			
tSa	Union		Remote Endpoint Address associated with the data. Only valid for datagram sockets (UDP). See Table 4.
abPayload	uint8_t [1472]		Payload Data of received datagram or stream.

Table 19: SOCK_CMD_RECVFROM_CNF – Packet

3.2.7 Send To service

The application can use the Send To service to send data to a remote endpoint via a socket.

3.2.7.1 SOCK_CMD_SENDTO_REQ Packet

The application shall send this packet to the stack in order to send data to a remote endpoint via a socket.

Packet structure reference

```
typedef struct SOCK_SENDTO_REQ_DATA_Ttag SOCK_SENDTO_REQ_DATA_T;

__PACKED_PRE struct __PACKED_POST SOCK_SENDTO_REQ_DATA_Ttag
{
  SOCK_H    hSocket;
  /** Reserved for future extensions set to zero */
  uint16_t     usFlags;
  /** remote address information */
  SOCK_ADDR_T  tSa;
  /** data */
  uint8_t      abBuffer[SOCK_LIMITS_MAX_PCK_SIZE];
};

typedef struct SOCK_SENDTO_REQ_Ttag SOCK_SENDTO_REQ_T;

__PACKED_PRE struct __PACKED_POST SOCK_SENDTO_REQ_Ttag
{
  SOCK_PCKHEADER_T       tHead;

  SOCK_SENDTO_REQ_DATA_T tData;
};
```

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination Queue-Handle.
ulLen	uint32_t	0x00000014 – 0x000005d4	Packet Data Length in bytes. Depends on amount of data to send.
ulCmd	uint32_t	0x0000961C	SOCK_CMD_SENDTO_REQ
Data			
hSocket	uint16_t		Handle of a socket to retrieve data from
usFlags	uint16_t	0	Receive Flags. Reserved for future. Set to zero.
tSa	Union		Remote Endpoint Address to send the data to. Only valid for datagram sockets. If set to zero for UDP Socket, a network broadcast will be performed according current IP address configuration. See Table 4 For more information about this structure.
abPayload	uint8_t [1472]		Payload Data of send datagram or stream.

Table 20: SOCK_CMD_SENDTO_REQ – Packet

3.2.7.2 SOCK_CMD_SENDTO_CNF packet

The stack will return this packet to the application for the SOCK_CMD_SENDTO_REQ packet when data was sent via the socket. If the socket is in Blocking Mode (default), the packet will not be returned until the socket is closed or data has been sent

Packet structure reference

```
typedef struct SOCK_EMPTY_PCK_Ttag SOCK_SENDTO_CNF_T;
```

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination Queue-Handle, unchanged
ulLen	uint32_t	0x000000000	Packet Data Length in bytes.
ulSta	uint32_t		See section *Error codes and status codes* on page 51.
ulCmd	uint32_t	0x0000961D	SOCK_CMD_SENDTO_CNF

Table 21: SOCK_CMD_SENDTO_CNF – Packet

3.2.8 Socket Close service

The application shall use the socket close service to close a socket. The service will close and destroy the associated socket object.

3.2.8.1 SOCK_CMD_CLOSE_REQ packet

The application shall send this packet to the stack in order to close a socket object.

Packet structure reference

```
typedef struct SOCK_CLOSE_REQ_DATA_Ttag SOCK_CLOSE_REQ_DATA_T;

__PACKED_PRE struct __PACKED_POST SOCK_CLOSE_REQ_DATA_Ttag
{
  /** Socket or file descriptor handle */
  SOCK_H         hSocket;
};


typedef struct SOCK_CLOSE_REQ_Ttag SOCK_CLOSE_REQ_T;

__PACKED_PRE struct SOCK_CLOSE_REQ_Ttag __PACKED_POST
{
  SOCK_PCKHEADER_T         tHead;

  SOCK_CLOSE_REQ_DATA_T tData;
};
```

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination Queue-Handle.
ulLen	uint32_t	0x00000002	Packet Data Length in bytes
ulCmd	uint32_t	0x00009602	SOCK_CMD_CLOSE_REQ
Data			
hSocket	uint16_t		Handle of File Descriptor to close

Table 22: SOCK_CMD_CLOSE_REQ – Packet

3.2.8.2 SOCK_CMD_CLOSE_CNF packet

The stack will return this packet to the application after executing the SOCK_CMD_CLOSE_REQ packet. The associated file or socket object has been destroyed, thus the socket handle is now invalid. Closing a TCP socket may take up to 120 seconds delay. This is necessary according TCP protocol specification in some states.

Packet structure reference

```
typedef struct SOCK_CLOSE_CNF_Ttag SOCK_CLOSE_CNF_T;
```

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination queue handle, unchanged
ulLen	uint32_t	0	Packet Data Length in bytes
ulSta	uint32_t		See section *Error codes and status codes* on page 51.
ulCmd	uint32_t	0x00009603	SOCK_CMD_CLOSE_CNF

Table 23: SOCK_CMD_CLOSE_CNF– Packet

3.2.9 Socket Abort service

The application shall use the socket abort service to close a socket immediately. This service is useful in the context of TCP connections. According to the TCP standard, a TCP socket must be held open in a special state for twice the Maximum Segment Lifetime after closing it. (2x 60 seconds). This is necessary to make sure that all pending incoming fragments are collected and acknowledged. During this time, the socket resource is still in use and cannot be reused by another connection. In several circumstances, it might be required to force a connection close without this timeout and immediately release the associated socket resource. This can be accomplished by using this service. The service will close the socket and release the socket resource immediately.

Note: This service is not a general replacement for the 3.2.8 Socket Close service. It might cause a violation of the TCP protocol and result in communication problems with network peers!

3.2.9.1 SOCK_CMD_ABORT_REQ packet

The application shall send this packet to the stack in order to abort a connection.

Packet structure reference

```
typedef struct SOCK_ABORT_REQ_DATA_Ttag SOCK_ABORT_REQ_DATA_T;

__PACKED_PRE struct __PACKED_POST SOCK_ABORT_REQ_DATA_Ttag
{
  SOCK_H         hSocket;
};

typedef struct SOCK_ABORT_REQ_Ttag SOCK_ABORT_REQ_T;

__PACKED_PRE struct __PACKED_POST SOCK_ABORT_REQ_Ttag
{
  SOCK_PCKHEADER_T      tHead;

  SOCK_ABORT_REQ_DATA_T tData;
};
```

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination Queue-Handle.
ulLen	uint32_t	0x00000002	Packet Data Length in bytes
ulCmd	uint32_t	0x0000961E	SOCK_CMD_ABORT_REQ
Data			
hSocket	uint16_t		Handle of File Descriptor to abort

Table 24: SOCK_CMD_ABORT_REQ – Packet

3.2.9.2 SOCK_CMD_ABORT_CNF packet

The stack will return this packet to the application after executing the SOCK_CMD_ABORT_REQ packet. Afterwards the associated file or socket object has been destroyed, thus the socket handle is invalid then.

Packet structure reference

```
typedef struct SOCK_EMPTY_PCK_Ttag SOCK_ABORT_CNF_T;
```

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination queue handle, unchanged
ulLen	uint32_t	0	Packet Data Length in bytes
ulSta	uint32_t		See section *Error codes and status codes* on page 51.
ulCmd	uint32_t	0x0000961F	SOCK_CMD_ABORT_CNF

Table 25: SOCK_CMD_ABORT_CNF– Packet

3.2.10 Socket Poll service

The application shall use the socket poll service to wait for an event on a socket. The service will return which file descriptors have pending events.

The service may be used in addition to be informed about IP address changes using the special file descriptor `SOCK_HANDLE_IPV4CHANGE 0xF000`.

3.2.10.1 SOCK_CMD_POLL_REQ packet

The application shall send this packet in order to wait for file descriptors. The length of the packet shall be set according the number of sockets to be polled. E.g. polling of two sockets results in 4 + 2 x 4 = 12 Bytes.

Packet structure reference

```
 typedef enum SOCK_POLL_EVENT_Etag SOCK_POLL_EVENT_E;

/** poll event bitmask, these are defined similar to posix values */
enum SOCK_POLL_EVENT_Etag
{
  SOCK_POLLIN   = 0x0001,
  SOCK_POLLPRI  = 0x0002,
  SOCK_POLLOUT  = 0x0004,
  SOCK_POLLERR  = 0x0008,
  SOCK_POLLHUP  = 0x0010,
  SOCK_POLLNVAL = 0x0020,
};

typedef struct SOCK_POLL_Ttag SOCK_POLL_T;

__PACKED_PRE struct __PACKED_POST SOCK_POLL_Ttag
{
  /** The file handle to poll */
  SOCK_H              hSocket;
  /** the events */
  uint16_t            usEventMsk;
};

typedef struct SOCK_POLL_REQ_DATA_Ttag SOCK_POLL_REQ_DATA_T;

__PACKED_PRE struct __PACKED_POST SOCK_POLL_REQ_DATA_Ttag
{
  int32_t            ilTimeout;
  /** array of file descriptors to poll
   *
   * actual array size might be smaller and must be specified in header */
  SOCK_POLL_T  atFds[SOCK_LIMITS_MAX_POLL];
};

typedef struct SOCK_POLL_REQ_Ttag SOCK_POLL_REQ_T;

__PACKED_PRE struct __PACKED_POST SOCK_POLL_REQ_Ttag
{
  SOCK_PCKHEADER_T          tHead;

  SOCK_POLL_REQ_DATA_T   tData;
};
```

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination Queue-Handle.
ulLen	uint32_t	0x00000004 - …	Packet Data Length in bytes. Depends on number of fds to poll
ulCmd	uint32_t	0x0000960A	SOCK_CMD_POLL_REQ
Data			
ilTimeout	INT32		Polling Timeout in milliseconds. Value 0 means to just check current events and immediate return. Negative Value means infinite wait until event occurs.
atFds[]	Array	X	Array of poll descriptor. One entry for each file descriptor to poll
hSocket	uint16_t		File descriptor handle
usEventMsk	uint16_t		Event Mask to specify for which events to wait. Multiple events should be combined by performing an OR-operation with the corresponding bitmasks to a single value. Some events are always enabled. (E.g. Errors)

Table 26: SOCK_CMD_POLL_REQ – Packet

Event Name	Value	Description
SOCK_POLLIN	0x0001	The socket is ready for reading data
SOCK_POLLPRI	0x0002	The socket has received high priority data and is ready for reading
SOCK_POLLOUT	0x0004	The socket is ready for writing data, or (depending on prior state) The connection was established successfully
SOCK_POLLERR	0x0008	The socket had an error (always active) If set, the error can be received with Get Socket Options service
SOCK_POLLHUP	0x0010	The remote peer has closed the connection (always active)
SOCK_POLLNVAL	0x0020	Invalid socket handle (always active)

Table 27: File descriptor poll event mask

3.2.10.2 SOCK_CMD_POLL_CNF packet

The stack will immediately return this packet to the application if any of the polled file descriptors has a pending event or a timeout of zero had been specified in the request. The packet will be returned after the specified timeout duration if no event occurred within the given interval.

Packet structure reference

```
typedef struct SOCK_POLL_CNF_DATA_Ttag SOCK_POLL_CNF_DATA_T;

__PACKED_PRE struct __PACKED_POST SOCK_POLL_CNF_DATA_Ttag
{
  /** number of fds with events */
  int32_t            ilNumFd;
  /** array of file descriptors to poll
   *
   * actual array size might be smaller and must be specified in header */
  SOCK_POLL_T  atFds[SOCK_LIMITS_MAX_POLL];
};


typedef struct SOCK_POLL_CNF_Ttag SOCK_POLL_CNF_T;

__PACKED_PRE struct __PACKED_POST SOCK_POLL_CNF_Ttag
{
  SOCK_PCKHEADER_T        tHead;

  SOCK_POLL_CNF_DATA_T tData;
};
```

Packet description

Variable		Type	Value / Range	Description
ulDest		uint32_t		Destination Queue-Handle, unchanged
ulLen		uint32_t		Packet Data Length in bytes, unchanged
ulSta		uint32_t	0	See section *Error codes and status codes* on page 51.
ulCmd		uint32_t	0x0000960B	SOCK_CMD_POLL_CNF
Data				
ilNumFd		INT32		Number of file descriptor where an event occurred
atFds[]		Array	X	Array of poll descriptor. One entry for each file descriptor to poll.
	hSocket	uint16_t		File descriptor handle, unchanged
	usEventMsk	uint16_t		Event Mask specifying which events occurred. Zero if no event occurred for the corresponding file descriptor

Table 28: SOCK_CMD_POLL_CNF – Packet

3.2.11 Socket Control service

The application shall use the socket control service to perform a specific control operation on a socket. For example, you can use it to switch a socket into non-blocking mode.

The service supports the following modes:

- Set/Get Non-blocking I/O mode
  - SOCK_CMD_RECVFROM_REQ: The request will not wait until data is received.
  - SOCK_CMD_SENDTO_REQ: The request will not wait until all data is send (data may be sent partially)
  - SOCK_CMD_ACCEPT_REQ: The request will not wait until client tries to connect
- Set/Get Non-blocking connect request
  - SOCK_CMD_CONNECT_REQ: The request will be responded immediately with status OK while the connection is established in background

Use the Poll service to check the status of the related socket.

3.2.11.1 SOCK_CMD_FCNTL_REQ packet

The application shall send this packet in order to perform a socket control.

Packet structure reference

```
typedef enum
{
  SOCK_FCNTL_GETFL = 3,
  SOCK_FCNTL_SETFL = 4,
} SOCK_FCNTL_E;

enum SOCK_STATUS_FLAGS_Etag
{
  /** Makes socket operations except connect non blocking */
  SOCK_FL_O_NONBLOCK = 0x0800,
  /** Makes socket operations including connect non blocking */
  SOCK_FL_O_NONBLOCK_CONNECT = 0x1000,
};
typedef enum SOCK_STATUS_FLAGS_Etag SOCK_STATUS_FLAGS_E;


__HIL_PACKED_PRE struct __HIL_PACKED_POST SOCK_FCNTL_COM_Ttag
{
  /** the file descriptor to perform the fcntl on */
  SOCK_H hSocket;
  /** the command to execute */
  uint16_t  usFcntl;
};

typedef struct SOCK_FCNTL_COM_Ttag SOCK_FCNTL_COM_T;

__HIL_PACKED_PRE struct __HIL_PACKED_POST SOCK_FCNTL_FL_Ttag
{
  /** the file descriptor to perform the fcntl on */
  SOCK_H hSocket;
  /** the command to execute */
  uint16_t  usFcntl;
  /** the status flags */
  uint32_t ulStatusFlags;
};
typedef struct SOCK_FCNTL_FL_Ttag SOCK_FCNTL_FL_T;

union SOCK_FCNTL_DATA_Ttag
{
  /** To access fields common to all fcntl commands */
  SOCK_FCNTL_COM_T tCom;
  /** used for F_GETFL/F_SETFL */
  SOCK_FCNTL_FL_T  tFileStatus;
};
typedef union SOCK_FCNTL_DATA_Ttag SOCK_FCNTL_DATA_T;
```

```
__HIL_PACKED_PRE struct __HIL_PACKED_POST SOCK_FCNTL_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  SOCK_FCNTL_DATA_T   tData;
};
typedef struct SOCK_FCNTL_REQ_Ttag SOCK_FCNTL_REQ_T;

__HIL_PACKED_PRE struct __HIL_PACKED_POST SOCK_FCNTL_CNF_Ttag
{
  HIL_PACKET_HEADER_T     tHead;

  SOCK_FCNTL_DATA_T tData;
};
typedef struct SOCK_FCNTL_CNF_Ttag SOCK_FCNTL_CNF_T;

union SOCK_FCNTL_PCK_Ttag
{
  SOCK_FCNTL_REQ_T tReq;
  SOCK_FCNTL_CNF_T tCnf;
};
typedef union SOCK_FCNTL_PCK_Ttag SOCK_FCNTL_PCK_T;
```

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination Queue-Handle.
ulLen	uint32_t	0x00000004 - …	Packet Data Length in bytes. Depends on the particular operation.
ulCmd	uint32_t	0x0000960C	SOCK_CMD_FCNTL_REQ
Data			
tCom	Struct	X	Union element describing the fields common to all fcntl operations
hSocket	uint16_t		File descriptor handle
usFcntl	uint16_t		Fcntl command code. See Table 30.
tFileStatus	Struct		Union element for use with Get/Set File Descriptor Status Flag command code
hSocket	uint16_t		File descriptor handle
usFcntl	uint16_t		Fcntl command code. See Table 30.
ulStatusFlags	uint32_t		File descriptor status flags bitmask. See Table 31.

Table 29: SOCK_CMD_FCNTL_REQ – Packet

Command Name	Value	Description
SOCK_FCNTL_GETFL	3	Get a file descriptor status flag
SOCK_FCNTL_SETFL	4	Set a file descriptor status flag

Table 30: File descriptor Fcntl command codes

Flag Name	Value	Description
SOCK_FL_O_NONBLOCK	0x0800	Non-blocking I/O Mode
SOCK_FL_O_NONBLOCK_CONNECT	0x1000	Non-blocking Connect request

Table 31: File descriptor Get/Set Status Flag bitmasks

3.2.11.2 SOCK_CMD_FCNTL_CNF packet

The stack will return this packet to the application after performing the file descriptor control operation.

Packet structure reference

```
typedef struct SOCK_FCNTL_CNF_Ttag SOCK_FCNTL_CNF_T;

__PACKED_PRE struct __PACKED_POST SOCK_FCNTL_CNF_Ttag
{
  SOCK_PCKHEADER_T     tHead;

  SOCK_FCNTL_DATA_T tData;
};
```

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination Queue-Handle, unchanged
ulLen	uint32_t		Packet Data Length in bytes, Depends on command and return code
ulSta	uint32_t	0	See section *Error codes and status codes* on page 51.
ulCmd	uint32_t	0x0000960D	SOCK_CMD_FCNTL_CNF
Data			
tCom	Struct	X	Union element describing the fields common to all fcntl operations
hSocket	uint16_t		File descriptor handle, unchanged
usFcntl	uint16_t		Fcntl command code. See Table 30, unchanged.
tFileStatus	Struct		Union element for use with Get/Set File Descriptor Status Flag command code
hSocket	uint16_t		File descriptor handle, unchanged
usFcntl	uint16_t		Fcntl command code. See Table 30, unchanged.
ulStatusFlags	uint32_t		File descriptor status flags bitmask. See Table 31.

Table 32: SOCK_CMD_FCNTL_CNF – Packet

3.2.12 Set Socket Options service

The application can use the set socket options service to configure socket options. Currently, the stack supports the following options for TCP sockets:

- Disable and enable the Naggle algorithm
- Disable and enable the KeepAlive
- Configure the timing values of KeepAlive
- Configure the linger timeout

3.2.12.1 SOCK_CMD_SETSOCKOPT_REQ packet

The application shall send this packet in order to configure a socket option

Packet structure reference

```
typedef struct SOCK_SETSOCKOPT_REQ_DATA_Ttag SOCK_SETSOCKOPT_REQ_DATA_T;

__PACKED_PRE struct __PACKED_POST SOCK_SETSOCKOPT_REQ_DATA_Ttag
{
  SOCK_H         hSocket;
  /** option level to set, value should be one of SOCK_IPPROTO_*  enums */
  uint16_t       usLevel;

  uint16_t       usOption;

  uint16_t       usReserved;
  /* valid union field depends on usOption */
  SOCK_SOCKOPT_T tOpt;
};

typedef struct SOCK_SETSOCKOPT_REQ_Ttag SOCK_SETSOCKOPT_REQ_T;

__PACKED_PRE struct __PACKED_POST SOCK_SETSOCKOPT_REQ_Ttag
{
  SOCK_PCKHEADER_T           tHead;

  SOCK_SETSOCKOPT_REQ_DATA_T tData;
};

typedef struct SOCK_EMPTY_PCK_Ttag SOCK_SETSOCKOPT_CNF_T;
```

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination Queue-Handle.
ulLen	uint32_t	0x00000010	Packet Data Length in bytes.
ulCmd	uint32_t	0x00009620	SOCK_CMD_SETSOCKOPT_REQ
Data			
hSocket	uint16_t		Socket Handle
usLevel	uint16_t	0, 1 or 6	Protocol Level , a value of SOCK_IPPROTO_IP, SOCK_IPPROTO_TCP or SOCK_SOL_SOCKET
usOption	uint16_t		The option to set, this is protocol specific. See section *Options structure definition* on page 48.
usReserved	uint16_t	0	Reserved set to zero
tOpt	UNION	X	
tOpt.tKeepAliveIntvl	STRUCT		Use in case of usOption SOCK_TCP_KEEPINTVL
iKeepIntvl	uint32_t	1 to 65535	The time in seconds between two non-acknowledged keep alive probes.
lReserved	int32_t	0	Reserved set to zero
tOpt.tKeepAliveIdle	STRUCT		Use in case of usOption SOCK_TCP_KEEPIDLE
lKeepIdle	uint32_t	1 to 65535	The time in seconds the connection needs to remain idle before starts sending keep alive probes
lReserved	int32_t	0	Reserved set to zero
tOpt.tKeepAliveCnt	STRUCT		Use in case of usOption SOCK_TCP_KEEPCNT
lKeepCnt	uint32_t	1 to 65535	Set to the number of keep alive probes to send before considering the connection dead.
lReserved	int32_t	0	Reserved set to zero
tOpt.tKeepAlive	STRUCT		Use in case of usOption SOCK_SO_KEEPALIVE
lKeepAlive	uint32_t	0,1	Set to 0 to disable the keepalive feature, set to 1 to enable the keepalive feature.
lReserved	int32_t	0	Reserved set to zero
tLinger	STRUCT		Use in case of usOption SOCK_SO_LINGER
lOnOff	INT32	0,1	Set to 0 to disable the linger feature, set to 1 to enable the linger feature.
lLinger	INT32	0 to 65535	Linger timeout in seconds
tOpt.tTcpNoDelay	STRUCT		Use in case of usOption SOCK_TCP_NODELAY
lEnabled	int32_t	0 or 1	Set to 0 to enable Naggle Algorithm (Default) and to 1 to disable Naggle Algorithm
lReserved	int32_t	0	Reserved set to zero
tOpt.tReuseAddr	STRUCT		Use in case of usOption SOCK_SO_REUSEADDR
lReuseAddr	Int32_t		Set to 1 to enable reusage of sockets even if their wait time is not expired.
lReserved	int32_t	0	Reserved set to zero
tOpt.tError	STRUCT		Use in case of usOption SOCK_SO_ERROR
ulError	uint32_t		
lReserved	int32_t	0	Reserved set to zero

Table 33: SOCK_SETSOCKOPT_REQ – Packet

3.2.12.2 SOCK_CMD_SETSOCKOPT_CNF packet

The stack will return this packet to the application after applying the socket option to the socket.

Packet structure reference

```
typedef struct SOCK_EMPTY_PCK_Ttag SOCK_SETSOCKOPT_CNF_T;
```

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination Queue-Handle, unchanged
ulLen	uint32_t	0	Packet Data Length in bytes, Depends on command and return code
ulSta	uint32_t	0	See section *Error codes and status codes* on page 51.
ulCmd	uint32_t	0x00009621	SOCK_CMD_SETSOCKOPT_CNF

Table 34: SOCK_CMD_SETSOCKOPT_CNF – Packet

3.2.13 Get Socket Options service

The application can use this service to retrieve the current setting for a particular socket option.

3.2.13.1 SOCK_CMD_GETSOCKOPT_REQ packet

The application shall use this packet to retrieve the current setting of a socket option.

Packet structure reference

```
typedef struct SOCK_GETSOCKOPT_REQ_DATA_Ttag SOCK_GETSOCKOPT_REQ_DATA_T;

__PACKED_PRE struct __PACKED_POST SOCK_GETSOCKOPT_REQ_DATA_Ttag
{
  SOCK_H         hSocket;

  uint16_t       usLevel;

  uint16_t       usOption;

  uint16_t       usReserved;
};

typedef struct SOCK_GETSOCKOPT_REQ_Ttag SOCK_GETSOCKOPT_REQ_T;

__PACKED_PRE struct __PACKED_POST SOCK_GETSOCKOPT_REQ_Ttag
{
  SOCK_PCKHEADER_T           tHead;

  SOCK_GETSOCKOPT_REQ_DATA_T tData;
};
```

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination Queue-Handle.
ulLen	uint32_t	0x00000008	Packet Data Length in bytes. Depends on the particular operation.
ulCmd	uint32_t	0x00009622	SOCK_CMD_GETSOCKOPT_REQ
Data			
hSocket	uint16_t		Socket Handle
usLevel	uint16_t	0, 1 or 6	Protocol Level , a value of SOCK_IPPROTO_IP, SOCK_IPPROTO_TCP or SOCK_SOL_SOCKET
usOption	uint16_t		The option to get, this is protocol specific. See section *Options structure definition* on page 48.
usReserved	uint16_t	0	Reserved

Table 35: SOCK_CMD_GETSOCKOPT_REQ – Packet

3.2.13.2 SOCK_CMD_GETSOCKOPT_CNF packet

The stack will return this packet to the application in response to the application's request.

Packet structure reference

```
typedef struct SOCK_GETSOCKOPT_CNF_DATA_Ttag SOCK_GETSOCKOPT_CNF_DATA_T;

/** Common fields of every sockopt data */
__PACKED_PRE struct __PACKED_POST SOCK_GETSOCKOPT_CNF_DATA_Ttag
{
  SOCK_H         hSocket;

  uint16_t       usLevel;

  uint16_t       usOption;

  uint16_t       usReserved;
  /* valid union field depends on usOption */
  SOCK_SOCKOPT_T tOpt;
};

typedef struct SOCK_GETSOCKOPT_CNF_Ttag SOCK_GETSOCKOPT_CNF_T;

__PACKED_PRE struct __PACKED_POST SOCK_GETSOCKOPT_CNF_Ttag
{
  SOCK_PCKHEADER_T           tHead;

  SOCK_GETSOCKOPT_CNF_DATA_T tData;
};
```

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination Queue-Handle.
ulLen	uint32_t	0, 0x10	Packet Data Length in bytes.
ulSta	uint32_t	0	See section *Error codes and status codes* on page 51.
ulCmd	uint32_t	0x00009623	SOCK_CMD_GETSOCKOPT_CNF
Data			
hSocket	uint16_t		Socket Handle, Unchanged from Request
usLevel	uint16_t		Unchanged from Request
usOption	uint16_t		Unchanged from Request
usReserved	uint16_t		Ignore
tOpt	UNION	X	See section *Options structure definition* on page 48 for details (depends on value of field usOption)
tOpt	UNION	X	
tOpt.tKeepAliveIntvl	STRUCT		Use in case of usOption SOCK_TCP_KEEPINTVL
iKeepIntvl	uint32_t		The time in seconds between two non-acknowledged keep alive probes.
lReserved	int32_t	0	Reserved, ignore.
tOpt.tKeepAliveIdle	STRUCT		Use in case of usOption SOCK_TCP_KEEPIDLE
lKeepIdle	uint32_t		The time in seconds the connection needs to remain idle before starts sending keep alive probes
lReserved	int32_t	0	Reserved, ignore.
tOpt.tKeepAliveCnt	STRUCT		Use in case of usOption SOCK_TCP_KEEPCNT
lKeepCnt	uint32_t	1 to 65535	The number of keep alive probes to send before considering the connection dead.
lReserved	int32_t	0	Reserved, ignore

tOpt.tKeepAlive	STRUCT		Use in case of usOption SOCK_SO_KEEPALIVE
lKeepAlive	uint32_t		0 indicates disabled keepalive feature, 1 indicates enabled keepalive feature.
lReserved	int32_t	0	Reserved, ignore.
tLinger	STRUCT		Use in case of usOption SOCK_SO_LINGER
lOnOff	INT32		0 indicates disabled linger feature, 1 indicates enabled linger feature.
lLinger	INT32	0 to 65535	Linger timeout in seconds (only valid if enabled)
tOpt.tTcpNoDelay	STRUCT		Use in case of usOption SOCK_TCP_NODELAY
lEnabled	int32_t	0 or 1	0 indicates enabled Naggle Algorithm, 1 indicates disabled Naggle Algorithm
lReserved	int32_t	0	Reserved, ignore.
tOpt.tReuseAddr	STRUCT		Use in case of usOption SOCK_SO_REUSEADDR
lReuseAddr	Int32_t		0 indicates disabled socket reusage feature, 1 indicates enabled reusage of sockets even if their wait time is not expired.
lReserved	int32_t	0	Reserved, ignore.
tOpt.tError	STRUCT		Use in case of usOption SOCK_SO_ERROR
ulError	uint32_t		Returns the recently occurred Socket Error code.
lReserved	int32_t	0	Reserved, ignore

Table 36: SOCK_CMD_GETSOCKOPT_CNF – Packet

3.2.14 Get Interface Addresses service

The application can use the get interface addresses service to retrieve the current interface addresses, e. g. the IP Address.

3.2.14.1 SOCK_GETIFADDRS_REQ packet

The application shall send this packet in order to obtain the interface addresses.

Packet structure reference

```
typedef struct SOCK_EMPTY_PCK_Ttag SOCK_GETIFADDRS_REQ_T;
```

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination Queue-Handle.
ulLen	uint32_t	0x00000000	Packet Data Length in bytes.
ulCmd	uint32_t	0x00009630	SOCK_CMD_GETIFADDRS_REQ

Table 37: SOCK_CMD_GETIFADDRS_REQ – Packet

3.2.14.2 SOCK_GETIFADDRS_CNF packet

The stack will return this packet to the application after receiving a SOCK_GETIFADDRS_REQ packet to retrieve interface addresses. The confirmation data is an array of address entries, the data length (tHead.ulLen) is used to indicate the number of entries. One interface (identified with the same name (tEntry.szIfName)) could be associated to several entries:

- If tEntry.tIfAddress.tCommon.bFamily is equal to SOCK_AF_INET, it is an IPv4 address. tNetmask and tBcastAddr are meaningful.
- If tEntry.tIfAddress.tCommon.bFamily is equal to SOCK_AF_PACKET, it is a layer 2 address (or hardware address). tNetmask and tBcastAddr are not meaningful.

Packet structure reference

```
typedef struct SOCK_EMPTY_PCK_Ttag SOCK_GETIFADDRS_REQ_T;

typedef struct SOCK_GETIFADDRS_ENTRY_Ttag SOCK_GETIFADDRS_ENTRY_T;

__HIL_PACKED_PRE struct __HIL_PACKED_POST SOCK_GETIFADDRS_ENTRY_Ttag
{
  uint8_t     szIfName[8];

  uint32_t    ulFlags;

  SOCK_ADDR_T tIfAddress;

  SOCK_ADDR_T tNetmask;

  SOCK_ADDR_T tBcastAddr;
};

typedef struct SOCK_GETIFADDRS_CNF_DATA_Ttag SOCK_GETIFADDRS_CNF_DATA_T;

__HIL_PACKED_PRE struct __HIL_PACKED_POST SOCK_GETIFADDRS_CNF_DATA_Ttag
{
```

```
#define SOCK_GETIFADDRS_MAX_NB_ENTRIES 8
  SOCK_GETIFADDRS_ENTRY_T atEntries[SOCK_GETIFADDRS_MAX_NB_ENTRIES];
};

typedef struct SOCK_GETIFADDRS_CNF_Ttag SOCK_GETIFADDRS_CNF_T;

__HIL_PACKED_PRE struct __HIL_PACKED_POST SOCK_GETIFADDRS_CNF_Ttag
{
  HIL_PACKET_HEADER_T        tHead;

  SOCK_GETIFADDRS_CNF_DATA_T tData;
};
```

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t		Destination Queue-Handle, unchanged
ulLen	uint32_t	0x00000039	Packet Data Length in bytes.
ulSta	uint32_t	0	See section *Error codes and status codes* on page 51.
ulCmd	uint32_t	0x00009631	SOCK_CMD_GETIFADDRS_CNF
Data			
atEntries	STRUCT[8]		Array of maximal 8 interface entries. One interface could be represented by several entries.

Table 38: SOCK_CMD_GETIFADDRS_CNF – Packet

Variable	Type	Value / Range	Description
szIfName	uint8_t[8]	“eth0”	Interface Name.
ulFlags	uint32_t		Ignore. Reserved for interface flags for future use.
tIpAddress	SOCK_ADDR_T		Address assigned to interface
tNetmask	SOCK_ADDR_T		Netmask assigned to interface, when the entry represents an IPv4 address.
tBcastAddr	SOCK_ADDR_T		Broadcast Address assigned to interface, when the entry represents an IPv4 address.

Table 39: SOCK_GETIFADDRS_ENTRY_T - Interface entry

3.3 Options structure definition

The *Set Socket Options service* (page 40) and the *Get Socket Options service* (page 43) use options listed in this section.

```
/** TCP socket options */
enum
{
  SOCK_TCP_NODELAY = 1,

  SOCK_TCP_KEEPCNT    = 0x05,
  SOCK_TCP_KEEPIDLE   = 0x06,
  SOCK_TCP_KEEPINTVL  = 0x07,
  SOCK_SO_KEEPALIVE   = 0x08,
  SOCK_SO_LINGER      = 0x09,
  SOCK_SO_REUSEADDR   = 0x0A,
  SOCK_SO_BROADCAST   = 0x0B,
  SOCK_SO_ERROR       = 0xF1,

};

/** Set sockopt tcp no delay */
typedef struct SOCK_SOCKOPT_TCP_NODELAY_Ttag SOCK_SOCKOPT_TCP_NODELAY_T;

__PACKED_PRE struct __PACKED_POST SOCK_SOCKOPT_TCP_NODELAY_Ttag
{
  /** Set to true if socket shall immediately send out the data */
  int32_t lEnabled;
  /** Padding to unique size. Set to zero in Set Sockopt Request,
   *  ignore in Get Sockopt Confirmation */
  int32_t  lReserved;
};

typedef struct SOCK_SOCKOPT_TCP_KEEPALIVE_INTVL_Ttag SOCK_SOCKOPT_TCP_KEEPALIVE_INTVL_T;
__PACKED_PRE struct __PACKED_POST SOCK_SOCKOPT_TCP_KEEPALIVE_INTVL_Ttag
{
  /* The time in seconds between two non-acknowledged keep alive probes. */
  int32_t lKeepIntvl;
  /** Padding to unique size. Set to zero in Set Sockopt Request,
   *  ignore in Get Sockopt Confirmation */
  int32_t  lReserved;
};

typedef struct SOCK_SOCKOPT_TCP_KEEPALIVE_IDLE_Ttag SOCK_SOCKOPT_TCP_KEEPALIVE_IDLE_T;
__PACKED_PRE struct __PACKED_POST SOCK_SOCKOPT_TCP_KEEPALIVE_IDLE_Ttag
{
  /* The time in seconds the connection needs to remain idle before starts sending keep
   * alive probes */
  int32_t lKeepIdle;
  /** Padding to unique size. Set to zero in Set Sockopt Request,
   *  ignore in Get Sockopt Confirmation */
  int32_t  lReserved;
};

typedef struct SOCK_SOCKOPT_TCP_KEEPALIVE_CNT_Ttag SOCK_SOCKOPT_TCP_KEEPALIVE_CNT_T;
__PACKED_PRE struct __PACKED_POST SOCK_SOCKOPT_TCP_KEEPALIVE_CNT_Ttag
{
  /* The number of keep alive probes to send before considering the connection dead */
  int32_t lKeepCnt;
  /** Padding to unique size. Set to zero in Set Sockopt Request,
   *  ignore in Get Sockopt Confirmation */
  int32_t  lReserved;
};

typedef struct SOCK_SOCKOPT_TCP_KEEPALIVE_Ttag SOCK_SOCKOPT_TCP_KEEPALIVE_T;
__PACKED_PRE struct __PACKED_POST SOCK_SOCKOPT_TCP_KEEPALIVE_Ttag
{
  /* Set to true (1) to enable keep alive or false (0) to disable it. */
  int32_t lKeepAlive;
  /** Padding to unique size. Set to zero in Set Sockopt Request,
```

```
   *  ignore in Get Sockopt Confirmation */
  int32_t  lReserved;
};

typedef struct SOCK_SOCKOPT_TCP_LINGER_Ttag SOCK_SOCKOPT_TCP_LINGER_T;
__PACKED_PRE struct __PACKED_POST SOCK_SOCKOPT_TCP_LINGER_Ttag
{
  /* Set to true (1) to enable linger feature or false (0) to disable it. */
  int32_t lOnOff;
  /* Linger time to be used in seconds (if enabled). */
  int32_t lLinger;
};

typedef struct SOCK_SOCKOPT_SO_REUSEADDR_Ttag SOCK_SOCKOPT_SO_REUSEADDR_T;
__HIL_PACKED_PRE struct __HIL_PACKED_POST SOCK_SOCKOPT_SO_REUSEADDR_Ttag
{
  /* Set to 1 to enable reusage of sockets even if their wait time is not expired. */
  int32_t  lReuseAddr;
  /** Padding to unique size. Set to zero in Set Sockopt Request,
   *  ignore in Get Sockopt Confirmation */
  int32_t  lReserved;
};

typedef struct SOCK_SOCKOPT_SO_ERROR_Ttag SOCK_SOCKOPT_SO_ERROR_T;
__PACKED_PRE struct __PACKED_POST SOCK_SOCKOPT_SO_ERROR_Ttag
{
  /* recently occurred Socket error. Can only be GET, not SET. */
  uint32_t ulError;
  /** Padding to unique size. Set to zero in Set Sockopt Request,
   *  ignore in Get Sockopt Confirmation */
  int32_t  lReserved;
};
```

```
typedef union SOCK_SOCKOPT_Ttag SOCK_SOCKOPT_T;

union SOCK_SOCKOPT_Ttag
{
  SOCK_SOCKOPT_TCP_NODELAY_T tTcpNoDelay;
  SOCK_SOCKOPT_TCP_KEEPALIVE_INTVL_T tKeepAliveIntvl;
  SOCK_SOCKOPT_TCP_KEEPALIVE_IDLE_T tKeepAliveIdle;
  SOCK_SOCKOPT_TCP_KEEPALIVE_CNT_T tKeepAliveCnt;
  SOCK_SOCKOPT_SO_KEEPALIVE_T tKeepAlive;
  SOCK_SOCKOPT_SO_LINGER_T tLinger;
  SOCK_SOCKOPT_SO_ERROR_T tError;
};
```

The following commands are available for the usOption value of the SetSockOpt and GetSockOpt request packets in order to select the option to access.

Command Name	Value	Description
SOCK_TCP_NODELAY	1	Disable or Enable Nagle Algorithm
SOCK_TCP_KEEPCNT	5	Set the number of keep alive probes send before considering the connection dead.
SOCK_TCP_KEEPIDLE	6	The time in seconds the connection needs to remain idle before starts sending keep alive probes
SOCK_TCP_KEEPINTVL	7	The time in seconds between two non-acknowledged keep alive probes.

Table 40: Socket options for protocol level SOCK_IPPROTO_IP and SOCK_IPPROTO_TCP

Command Name	Value	Description
SOCK_SO_KEEPALIVE	8	Enable/disable the keep alive feature.
SOCK_SO_LINGER	9	Configure the linger timeout, in order to set the maximum time of TIME-WAIT state. After timeout, the socket resources are freed and the close packet confirmation will be send. **Remark**: The on/off Boolean is in Socket API not relevant. After a close packet has been transmitted, it always “lingers” before transmitting the closing confirmation packet
SOCK_SO_ERROR	0xF1	Get the last error code.

Table 41: Socket options for protocol level SOCK_SOL_SOCKET

4 Error codes and status codes

The following status and error codes are used by the Socket interface component in `ulSta`.

Hexadecimal Value	Definition Description
0x00000000	SUCCESS_HIL_OK Status ok.
0xC0C90001	ERR_SOCK_UNSUPPORTED_SOCKET Unsupported socket domain, type and protocol combination.
0xC0C90002	ERR_SOCK_INVALID_SOCKET_HANDLE Invalid socket handle.
0xC0C90003	ERR_SOCK_SOCKET_CLOSED Socket was closed.
0xC0C90004	ERR_SOCK_INVALID_OP The command is invalid for the particular socket.
0xC0C90005	ERR_SOCK_INVALID_ADDRESS_FAMILY An invalid address family was used for this socket
0xC0C90006	ERR_SOCK_IN_USE The specified address is already in use.
0xC0C90007	ERR_SOCK_HUP The remote side closed the connection
0xC0C90008	ERR_SOCK_WOULDBLOCK The operation would block
0xC0C90009	ERR_SOCK_ROUTE No IP route to destination address.
0xC0C9000A	ERR_SOCK_IS_CONNECTED IP socket already connected.
0xC0C9000B	ERR_SOCK_CONNECTION_ABORTED TCP connection aborted.
0xC0C9000C	ERR_SOCK_CONNECTION_RESET TCP connection reset.
0xC0C9000D	ERR_SOCK_CONNECTION_CLOSED TCP connection closed.
0xC0C9000E	ERR_SOCK_NOT_CONNECTED IP Socket not connected.
0xC0C9000F	ERR_SOCK_NETWORK_INTERFACE Low-level network interface error.

Table 42: Socket API error codes

5 Appendix

5.1 List of tables

5.2 Legal notes

Copyright

Important notes

Utmost care was/is given in the preparation of the documentation at hand consisting of a user's manual, operating manual and any other document type and accompanying texts. However, errors cannot be ruled out. Therefore, we cannot assume any guarantee or legal responsibility for erroneous information or liability of any kind. You are hereby made aware that descriptions found in the user's manual, the accompanying texts and the documentation neither represent a guarantee nor any indication on proper use as stipulated in the agreement or a promised attribute. It cannot be ruled out that the user's manual, the accompanying texts and the documentation do not completely match the described attributes, standards or any other data for the delivered product. A warranty or guarantee with respect to the correctness or accuracy of the information is not assumed.

We reserve the right to modify our products and the specifications for such as well as the corresponding documentation in the form of a user's manual, operating manual and/or any other document types and accompanying texts at any time and without notice without being required to notify of said modification. Changes shall be taken into account in future manuals and do not represent an obligation of any kind, in particular there shall be no right to have delivered documents revised. The manual delivered with the product shall apply.

Under no circumstances shall Hilscher Gesellschaft für Systemautomation mbH be liable for direct, indirect, ancillary or subsequent damage, or for any loss of income, which may arise after use of the information contained herein.

Liability disclaimer

The hardware and/or software was created and tested by Hilscher Gesellschaft für Systemautomation mbH with utmost care and is made available as is. No warranty can be assumed for the performance or flawlessness of the hardware and/or software under all application conditions and scenarios and the work results achieved by the user when using the hardware and/or software. Liability for any damage that may have occurred as a result of using the hardware and/or software or the corresponding documents shall be limited to an event involving willful intent or a grossly negligent violation of a fundamental contractual obligation. However, the right to assert damages due to a violation of a fundamental contractual obligation shall be limited to contract-typical foreseeable damage.

It is hereby expressly agreed upon in particular that any use or utilization of the hardware and/or software in connection with

- Flight control systems in aviation and aerospace;
- Nuclear fission processes in nuclear power plants;
- Medical devices used for life support and
- Vehicle control systems used in passenger transport

shall be excluded. Use of the hardware and/or software in any of the following areas is strictly prohibited:

- For military purposes or in weaponry;
- For designing, engineering, maintaining or operating nuclear systems;
- In flight safety systems, aviation and flight telecommunications systems;
- In life-support systems;
- In systems in which any malfunction in the hardware and/or software may result in physical injuries or fatalities.

You are hereby made aware that the hardware and/or software was not created for use in hazardous environments, which require fail-safe control mechanisms. Use of the hardware and/or software in this kind of environment shall be at your own risk; any liability for damage or loss due to impermissible use shall be excluded.

Warranty

Hilscher Gesellschaft für Systemautomation mbH hereby guarantees that the software shall run without errors in accordance with the requirements listed in the specifications and that there were no defects on the date of acceptance. The warranty period shall be 12 months commencing as of the date of acceptance or purchase (with express declaration or implied, by customer's conclusive behavior, e.g. putting into operation permanently).

The warranty obligation for equipment (hardware) we produce is 36 months, calculated as of the date of delivery ex works. The aforementioned provisions shall not apply if longer warranty periods are mandatory by law pursuant to Section 438 (1.2) BGB, Section 479 (1) BGB and Section 634a (1) BGB [Bürgerliches Gesetzbuch; German Civil Code] If, despite of all due care taken, the delivered product should have a defect, which already existed at the time of the transfer of risk, it shall be at our discretion to either repair the product or to deliver a replacement product, subject to timely notification of defect.

The warranty obligation shall not apply if the notification of defect is not asserted promptly, if the purchaser or third party has tampered with the products, if the defect is the result of natural wear, was caused by unfavorable operating conditions or is due to violations against our operating regulations or against rules of good electrical engineering practice, or if our request to return the defective object is not promptly complied with.

Costs of support, maintenance, customization and product care

Please be advised that any subsequent improvement shall only be free of charge if a defect is found. Any form of technical support, maintenance and customization is not a warranty service, but instead shall be charged extra.

Additional guarantees

Although the hardware and software was developed and tested in-depth with greatest care, Hilscher Gesellschaft für Systemautomation mbH shall not assume any guarantee for the suitability thereof for any purpose that was not confirmed in writing. No guarantee can be granted whereby the hardware and software satisfies your requirements, or the use of the hardware and/or software is uninterruptable or the hardware and/or software is fault-free.

It cannot be guaranteed that patents and/or ownership privileges have not been infringed upon or violated or that the products are free from third-party influence. No additional guarantees or promises shall be made as to whether the product is market current, free from deficiency in title, or can be integrated or is usable for specific purposes, unless such guarantees or promises are required under existing law and cannot be restricted.

Confidentiality

The customer hereby expressly acknowledges that this document contains trade secrets, information protected by copyright and other patent and ownership privileges as well as any related rights of Hilscher Gesellschaft für Systemautomation mbH. The customer agrees to treat as confidential all of the information made available to customer by Hilscher Gesellschaft für Systemautomation mbH and rights, which were disclosed by Hilscher Gesellschaft für Systemautomation mbH and that were made accessible as well as the terms and conditions of this agreement itself.

The parties hereby agree to one another that the information that each party receives from the other party respectively is and shall remain the intellectual property of said other party, unless provided for otherwise in a contractual agreement.

The customer must not allow any third party to become knowledgeable of this expertise and shall only provide knowledge thereof to authorized users as appropriate and necessary. Companies associated with the customer shall not be deemed third parties. The customer must obligate authorized users to confidentiality. The customer should only use the confidential information in connection with the performances specified in this agreement.

The customer must not use this confidential information to his own advantage or for his own purposes or rather to the advantage or for the purpose of a third party, nor must it be used for commercial purposes and this confidential information must only be used to the extent provided for in this agreement or otherwise to the extent as expressly authorized by the disclosing party in written form. The customer has the right, subject to the obligation to confidentiality, to disclose the terms and conditions of this agreement directly to his legal and financial consultants as would be required for the customer's normal business operation.

Export provisions

The delivered product (including technical data) is subject to the legal export and/or import laws as well as any associated regulations of various countries, especially such laws applicable in Germany and in the United States. The products / hardware / software must not be exported into such countries for which export is prohibited under US American export control laws and its supplementary provisions. You hereby agree to strictly follow the regulations and to yourself be responsible for observing them. You are hereby made aware that you may be required to obtain governmental approval to export, reexport or import the product.

5.3 Contacts

Headquarters

Germany
Hilscher Gesellschaft für
Systemautomation mbH
Rheinstrasse 15
65795 Hattersheim
Phone: +49 (0) 6190 9907-0
Fax: +49 (0) 6190 9907-50
E-Mail: info@hilscher.com

Support
Phone: +49 (0) 6190 9907-99
E-Mail: de.support@hilscher.com

Subsidiaries

China
Hilscher Systemautomation (Shanghai) Co. Ltd.
200010 Shanghai
Phone: +86 (0) 21-6355-5161
E-Mail: info@hilscher.cn

Support
Phone: +86 (0) 21-6355-5161
E-Mail: cn.support@hilscher.com

France
Hilscher France S.a.r.l.
69800 Saint Priest
Phone: +33 (0) 4 72 37 98 40
E-Mail: info@hilscher.fr

Support
Phone: +33 (0) 4 72 37 98 40
E-Mail: fr.support@hilscher.com

India
Hilscher India Pvt. Ltd.
Pune, Delhi, Mumbai
Phone: +91 8888 750 777
E-Mail: info@hilscher.in

Italy
Hilscher Italia S.r.l.
20090 Vimodrone (MI)
Phone: +39 02 25007068
E-Mail: info@hilscher.it

Support
Phone: +39 02 25007068
E-Mail: it.support@hilscher.com

Japan
Hilscher Japan KK
Tokyo, 160-0022
Phone: +81 (0) 3-5362-0521
E-Mail: info@hilscher.jp

Support
Phone: +81 (0) 3-5362-0521
E-Mail: jp.support@hilscher.com

Korea
Hilscher Korea Inc.
Seongnam, Gyeonggi, 463-400
Phone: +82 (0) 31-789-3715
E-Mail: info@hilscher.kr

Switzerland
Hilscher Swiss GmbH
4500 Solothurn
Phone: +41 (0) 32 623 6633
E-Mail: info@hilscher.ch

Support
Phone: +49 (0) 6190 9907-99
E-Mail: ch.support@hilscher.com

USA
Hilscher North America, Inc.
Lisle, IL 60532
Phone: +1 630-505-5301
E-Mail: info@hilscher.us

Support
Phone: +1 630-505-5301
E-Mail: us.support@hilscher.com