Function block library

IP_Com_8

for PLCnext Engineer

Documentation for
PHOENIX CONTACT function blocks
PHOENIX CONTACT GmbH Co. KG
Flachsmarktstrasse 8
D-32825 Blomberg, Germany

This documentation is available in English only.

Table of Contents

1 Installation hint

If you did not specify a different directory during **library** installation all data in the MSI file will be unpacked to

c:\Users\Public\Documents\Phoenix Contact Libraries\PLCnext Engineer (former: PC Worx Engineer)

Please copy the library data to your PLCnext Engineer (former: PC Worx Engineer) working library directory.

If you did not specify a different directory during **PLCnext Engineer** installation the default PLCnext Engineer working library directory is

c:\Users\Public\Documents\PLCnext Engineer\Libraries (former: PC Worx Engineer\Libraries)

2 General information

This function block library contains function blocks for establishing IP connections via a controller. The function blocks make it possible to establish multiple successive connections to different network devices with one instance of the firmware function block required for IP connections (TCP_SOCKET, TCP_SEND, TCP_RECEIVE, UDP_SOCKET, UDP_SEND, UDP_RECEIVE, TLS_SOCKET, TLS_SEND, TLS_RECEIVE). This means that, out of all the available instantiations of the firmware function block (number is device-specific), only one instance is occupied for multiple IP connections.

The mode is determined via the variables xUDP and xTLS. UDP mode has highest priority here.

Independent IP connections can be established with the function blocks in this library. The IPC_Socket function block is still required for the use of additional function block libraries in order to enable IP communication.

For server functionalities and connections maintained over prolonged periods of time, the use of the IPC_Socket is not recommended. In such cases, the firmware function blocks should be used. For more information on the function of the firmware function blocks, refer to the PC Worx Engineer help documentation.

3 Change notes

Library version	Library build	PLCnext Engineer version	Change notes	Supported PLCs
8	20200204	>= 2020.0 LTS	Bug fix: Reset x_EN_R from firmwareblock UDP/TCP/TLS_Receive on disconnection	AXC F 1152 (1151412) AXC F 2152 (2404267)
7	20191029	2019.0 LTS 2019.3 2019.6 2019.9	Adaptation to TLS function blocks	AXC F 2152 (2404267)
6	20191010	2019.0 LTS 2019.3 2019.6 2019.9	Revised documentation	AXC F 2152 (2404267)
6	20190926	2019.0 LTS 2019.3 2019.6 2019.9	Added variables to structure: • uiBind_Port • strBind_IP • strSource_IP • uiSource_Port Adapted to PLCnext Engineer 2019.9	AXC F 2152 (2404267)
5	20190717	2019.0 LTS 2019.3 2019.6	Adapted to PLCnext Engineer 2019.6	AXC F 2152 (2404267)
4	20190607	2019.0 LTS	IPC_Socket_2: • Input xActivate moved to udtIP_Socket • Added TLS_* function blocks • Bug fixes • Added auto retry	AXC F 2152 (2404267)
3	20190305	2019.0 LTS	Adapted to PLCnext Engineer 2019.0 LTS	AXC F 2152 (2404267)
2	20190208	PCWE 7.2.1	Added xOccupied	AXC F 2152 (2404267)
1	20180306	PCWE 7.2.1	Converted from PC Worx 6	AXC F 2152 (2404267)

New version number: Functional changes of at least one function block, incompatibilities (e.g. change of library format)

New build number: No functional changes, but changes in the MSI file (e.g. documentation update, additional examples)

Note: This library is only released for a cycle time of 20 ms or more.

4 Function blocks

Function block	Description	Version	Supported articles	License
IPC_Socket	This function block manages the socket firmware function blocks	5	-	none
IPC_DiagInfo_EN	This function block outputs the last diagnostics of the connection as English text	1	-	none

5 IPC_Socket

This function block manages the socket (firmware function blocks TCP/UDP/TLS_SOCKET, TCP/UDP/TLS_SEND, and TCP/UDP/TLS_RECEIVE) for the connected IP connections (clients connected via the structure udtIP_Socket). This function block only has to be instantiated once for the connected clients. If there are multiple instances of IPC_Socket (allocation of clients) the startup instructions must be observed.

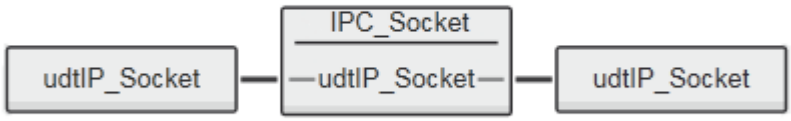
If a falling edge is detected in xSendRequest of the udtIP_Socket, the IPC_Socket does not terminate the connection. If a falling edge is detected in xConnect of the udtIP_Socket, the IPC_Socket does terminate the connection. Before a connection is established, the variables xTLS, xUDP and xIS_SRV must be set. If udtIP_Socket.xAutoReset is TRUE, the IPC_Socket tries to reconnect after an error has occurred. A connection is established on a rising edge at xConnect or xSendRequest if the udtIP_Socket.xOccupied is FALSE. If xOccupied is TRUE, the IPC_Socket waits until xOccupied is FALSE and then establishes the connection. As long as xBusy is TRUE do not change the IP parameter. As long as a valid connection exists the mode (UDP/TCP/TLS) may not be changed.

All parameters must be reset before the IPC_Socket can be used.

For details refer to TCP/UDP/TLS_* documentation.

The IP connection to the client is maintained. In the event of a transmission/connection request to a different client, this existing connection will be terminated.

5.1 Function block call



5.2 Input parameters

None

5.3 Output parameters

None

5.4 Inout parameters

Name	Type	Description
udtIP_Socket	IPC_UDT_IP_SOCKET	Structure for communication between IPC_Socket and other function blocks.

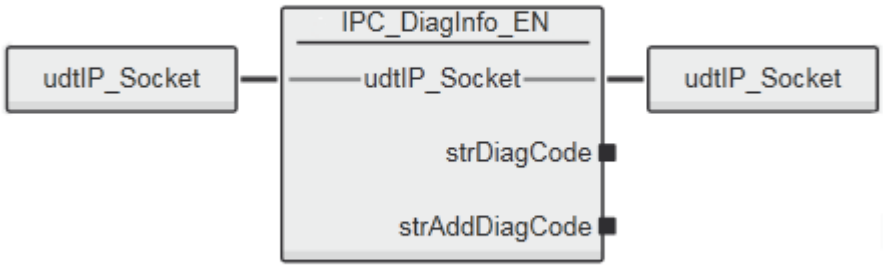
5.5 Diagnosis

DiagCode	AddDiagCode	Meaning
16#0000	16#0000	Block is not activated
16#8000	16#0000	Block is active
16#C330		Error of the TCP/UDP/TLS_SOCKET firmware function block
16#C330	16#XXXX	Status of the TCP/UDP/TLS_SOCKET firmware function block
16#C331		Error of the TCP/UDP/TLS_RECEIVE firmware function block
16#C331	16#XXXX	Status of the TCP/UDP/TLS_RECEIVE firmware function block
16#C332		Error of the TCP/UDP/TLS_SEND firmware function block
16#C332	16#XXXX	Status of the TCP/UDP/TLS_SEND firmware function block
16#C110	16#0001	Wrong input parameter (udiDataCNT)
16#C410	16#0000	Connect timeout
16#C414	16#0000	Sending timeout

6 IPC_DiagInfo_EN

This function block outputs the last diagnostics of the connection as English text, as long as the IPC_Socket is activated.

6.1 Function block call



6.2 Input parameters

None

6.3 Output parameters

Name	Type	Description
strDiagCode	STRING	The output indicates the diagnostics of the IPC_SOCKET function block as English text
strAddDiagCode	STRING	The output indicates the extended diagnostics of the IPC_SOCKET function block as English text

6.4 Inout parameters

Name	Type	Description
udtIP_Socket	IPC_UDT_IP_SOCKET	Structure for communication between IPC_Socket and other function blocks.

6.5 Diagnosis

This function block has no diagnosis

7 Startup examples

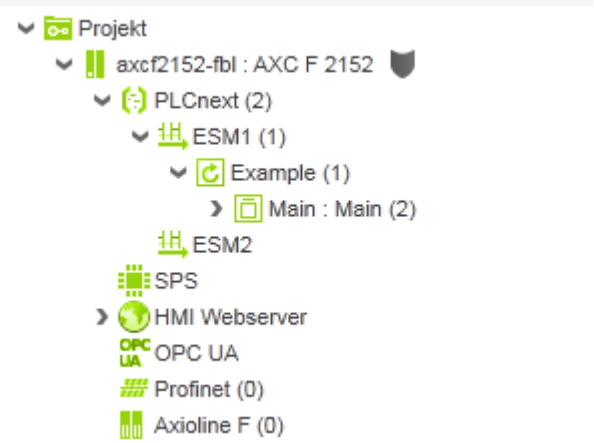
For the startup instruction of the IPC_Socket function block please find the following example:

IPC_8_EXA_IPC_SOCKET.pcwex

Plant

For this example, the following hardware is used:

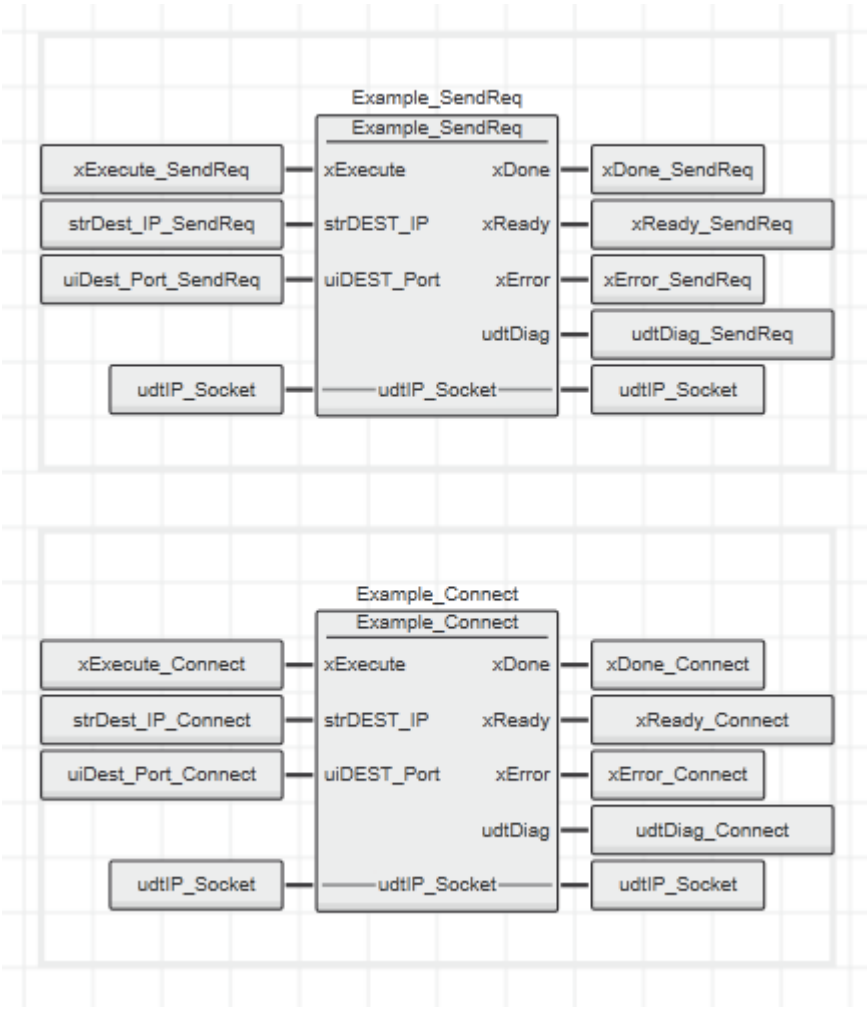
- AXC F 2152 (2404267)



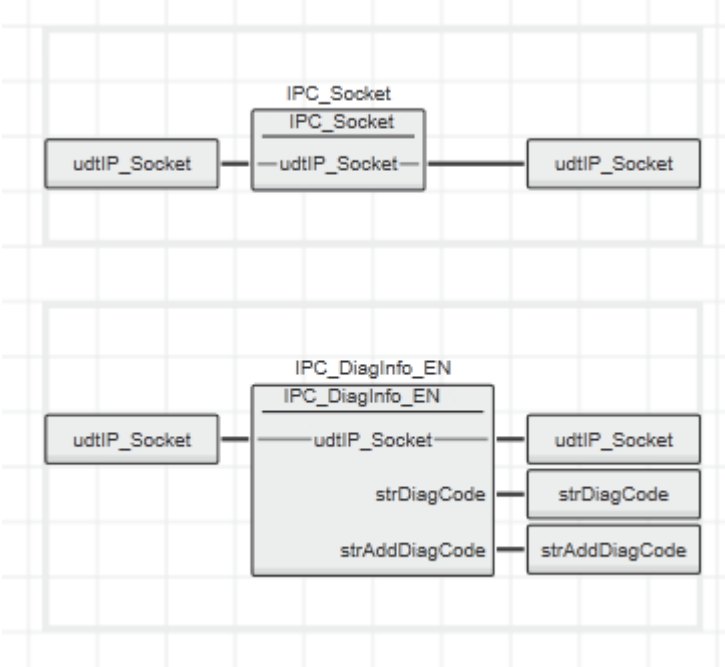
This example is located in the “Examples” folder of the unzipped msi file of the library. It describes the use of the IPC_Socket function block.

The Example contains two function blocks. The one shows the behavior with xSendReq the other one shows the behavior with xConnect.

7.1 Example FB's



7.2 IPC_Socket



7.3 Example_SendReq

```
CASE iState OF
    0: (*wait for start*)
        udtDiag.wDiagCode       :=  16#0000;
        udtDiag.wAddDiagCode    :=  16#0000;
        xError                  :=  FALSE;
        xReady                  :=  TRUE;
        IF xExecute = TRUE THEN (*start execution of function block*)
            FOR iIdx := 0 TO 99 DO (*add data to send array*)
                udtIP_Socket.udtSend.udtSndData.arrData[iIdx] := TO_BYTE(iIdx);
            END_FOR;
            xReady  :=  FALSE;
            iState  :=  10;
        END_IF;
    10: (*check status of udtIP_Socket.xOccupied*)
        IF udtIP_Socket.xOccupied = FALSE THEN
        (*set xOccupied = TRUE to block the udtIP_Socket structure*)
            udtIP_Socket.xOccupied  :=  TRUE;
            udtDiag.wDiagCode       :=  16#8000;
            iState                  :=  20;
        ELSE
        (*udtIP_Socket is used by another function block
        wait until xOccupied = FALSE*)
            udtDiag.wDiagCode       :=  WORD#8500;
            udtDiag.wAddDiagCode    :=  WORD#0010;
        END_IF;
    20: (*activate the IPC_SOCKET function block*)
        IF udtIP_Socket.xActive = FALSE THEN
            udtIP_Socket.xActivate  :=  TRUE;
        ELSIF udtIP_Socket.xActive = TRUE THEN
        (*IPC_SOCKET is already activated*)
            iState  := 25;
        END_IF;
    25: (*init the IPC_Socket function block &
        set all variables*)
        udtIP_Socket.xUDP   :=  FALSE;
        udtIP_Socket.xTLS   :=  FALSE;
        udtIP_Socket.udtConnect.xIS_SRV     :=  FALSE;
        udtIP_Socket.udtConnect.strDEST_IP  :=  strDEST_IP;
        udtIP_Socket.udtConnect.uiDEST_Port :=  uiDEST_Port;
        udtIP_Socket.udtSend.udiDataCNT     :=  100;
        iState                              :=  30;
    30: (*connect with the partner and send data*)
        udtIP_Socket.udtSend.xSendReq   :=  TRUE;
        iState  :=  40;
    40: (*wait for udtSend.xDone*)
        IF (
            udtIP_Socket.udtSend.xDone = TRUE (*for one cycle*)
            AND udtIP_Socket.udtConnect.xActive = TRUE
            AND udtIP_Socket.udtConnect.xConnected = TRUE
        ) THEN
            xDone   :=  TRUE; (*just for one cycle*)
            udtIP_Socket.udtSend.xSendReq   :=  FALSE; (*reset xSendReq*)
            iState  :=  50;
        ELSIF udtIP_Socket.xError = TRUE THEN
        (*(*error while sending*)*)
            udtDiag.wDiagCode       :=  udtIP_Socket.udtDiag.wDiagCode;
            udtDiag.wAddDiagCode    :=  udtIP_Socket.udtDiag.wAddDiagCode;
```

```
            iState  :=  99; (*go to error state*)
        END_IF;
    50: (*set xOccupied = FALSE that other function blocks can use the structure
        udtIP_Socket.xOccupied          :=  FALSE;
        xDone   :=  FALSE;
        iState  :=  60;

    60: (*wait for xExecute = FALSE*)
        IF xExecute = FALSE THEN
            iState  :=  0;
        END_IF;
    99: (*error wait for xExecute = FALSE*)
        IF xExecute = FALSE THEN
        (*in case of error xOccupied stays TRUE until xExecute is set FALSE*)
            udtIP_Socket.xActivate  :=  FALSE;
            udtIP_Socket.xOccupied  :=  FALSE;
            iState  :=  0;
        END_IF;
END_CASE;
```

7.4 Example_Connect

```
CASE iState OF
    0: (*wait for start*)
        udtDiag.wDiagCode       :=  16#0000;
        udtDiag.wAddDiagCode    :=  16#0000;
        xError                  :=  FALSE;
        xReady                  :=  TRUE;
        IF xExecute = TRUE THEN (*start execution of function block*)
            FOR iIdx := 0 TO 99 DO (*add data to send array*)
                udtIP_Socket.udtSend.udtSndData.arrData[iIdx] := TO_BYTE(iIdx);
            END_FOR;
            xReady  :=  FALSE;
            iState  :=  10;
        END_IF;
    10: (*check status of udtIP_Socket.xOccupied*)
        IF udtIP_Socket.xOccupied = FALSE THEN
        (*set xOccupied = TRUE to block the udtIP_Socket structure*)
            udtIP_Socket.xOccupied  :=  TRUE;
            udtDiag.wDiagCode       :=  16#8000;
            iState                  :=  20;
        ELSE
        (*udtIP_Socket is used by another function block
        wait until xOccupied = FALSE*)
            udtDiag.wDiagCode       :=  WORD#8500;
            udtDiag.wAddDiagCode    :=  WORD#0010;
        END_IF;
    20: (*activate the IPC_SOCKET function block*)
        IF udtIP_Socket.xActive = FALSE THEN
            udtIP_Socket.xActivate  :=  TRUE;
        ELSIF udtIP_Socket.xActive = TRUE THEN
        (*IPC_SOCKET is already activated*)
            iState  := 25;
        END_IF;
    25: (*init the IPC_Socket function block &
        set all variables*)
        udtIP_Socket.xUDP   :=  FALSE;
        udtIP_Socket.xTLS   :=  FALSE;
        udtIP_Socket.udtConnect.xIS_SRV     :=  FALSE;
        udtIP_Socket.udtConnect.strDEST_IP  :=  strDEST_IP;
        udtIP_Socket.udtConnect.uiDEST_Port :=  uiDEST_Port;
        udtIP_Socket.udtSend.udiDataCNT     :=  100;
        iState                              :=  30;
    30: (*connect with the partner and send data*)
        IF udtIP_Socket.udtConnect.xConnect = FALSE THEN
            iState  := 35;
        ELSE
            udtIP_Socket.udtConnect.xConnect    :=  FALSE;
            iState  := 35;
        END_IF;
    35:
        IF udtIP_Socket.udtConnect.xConnected = FALSE THEN
            udtIP_Socket.udtConnect.xConnect    :=  TRUE;
        END_IF;
        IF (
            udtIP_Socket.udtConnect.xActive = TRUE
            AND udtIP_Socket.udtConnect.xConnected = TRUE
        ) THEN
            xConnected  := TRUE;
```

```
            iState  :=  40;
        ELSIF udtIP_Socket.xError = TRUE THEN
        (*error establishing the connection *)
            xError  := TRUE;
            udtDiag.wDiagCode       :=  udtIP_Socket.udtDiag.wDiagCode;
            udtDiag.wAddDiagCode    :=  udtIP_Socket.udtDiag.wAddDiagCode;
            iState  :=  99; (*go to error state*)
        END_IF;
    40: (*start send data*)
        udtIP_Socket.udtSend.xSendReq   :=  TRUE;
        iState  :=  50;

    50: (*wait for udtSend.xDone*)
        IF (
            udtIP_Socket.udtSend.xDone = TRUE
            AND udtIP_Socket.udtConnect.xActive = TRUE
            AND udtIP_Socket.udtConnect.xConnected = TRUE
        ) THEN
            xDone   :=  TRUE; (*TRUE for one cycle*)
            udtIP_Socket.udtSend.xSendReq   :=  FALSE; (*reset xSendReq*)
            iState  :=  60;
        ELSIF udtIP_Socket.xError = TRUE THEN
        (*error while sending*)
            xError  := TRUE;
            udtDiag.wDiagCode       :=  udtIP_Socket.udtDiag.wDiagCode;
            udtDiag.wAddDiagCode    :=  udtIP_Socket.udtDiag.wAddDiagCode;
            iState  :=  99; (*go to error state*)
        END_IF;
    60: (*set xOccupied = FALSE that other function blocks can use the structure
        udtIP_Socket.xOccupied          :=  FALSE;
        xDone   :=  FALSE;
        iState  :=  70;

    70: (*wait for xExecute = FALSE*)
        IF xExecute = FALSE THEN
            iState  :=  0;
        END_IF;
    99: (*error wait for xExecute = FALSE*)
        IF xExecute = FALSE THEN
        (*in case of error xOccupied stays TRUE until xExecute is set FALSE*)
            udtIP_Socket.xActivate  :=  FALSE;
            udtIP_Socket.xOccupied  :=  FALSE;
            iState  :=  0;
        END_IF;
END_CASE;
```

8 Appendix

8.1 TCP/UDP/TLS_*

ERROR = FALSE

Status code	Description
16#0000	Situation is normal (no error).
16#8000	Socket is trying to connect the partner.
16#8001	Server is listening for a client.
16#8002	Server has rejected a client because the IP address and port number do not match.
16#8003	Not all data could be sent. Remaining data will be sent in the next cycle(s).
16#8004	Not all data received: Received length < Expected length

ERROR = TRUE

Error code	Description	Error only for
16#C001	Socket creation failed.	
16#C002	IP has wrong format.	
16#C003	Memory allocation failed.	
16#C100	Unexpected error during connecting of a client to a server.	TCP/TLS_SOCKET
16#C101	Unexpected error during receive operation.	UDP/TCP/TLS_RECEIVE
16#C102	Unexpected error during send operation.	UDP/TCP/TLS_SEND
16#C103	Unexpected error during bind operation.	UDP_SOCKET
16#C104	Unexpected error during listen operation.	TCP/TLS_SOCKET
16#C105	Unexpected error during accept operation.	TCP/TLS_SOCKET
16#C150	The TLS parameterization of the TLS_SEND/TLS_RECEIVE function blocks is inconsistent with the TLS_SOCKET function block. This is the case when: • TLS_SEND/TLS_RECEIVE require secure transmission/reception of data (SEND_SECURE/RECEIVE_SECURE input = TRUE), but the socket is not yet initialized for TLS communication (START_TLS input of TLS_SOCKET is FALSE). • TLS_SEND/TLS_RECEIVE require insecure transmission/reception of data (SEND_SECURE/RECEIVE_SECURE input = FALSE), but the socket is already initialized for TLS communication (START_TLS input of TLS_SOCKET is TRUE).	TLS_*
16#C151	An error regarding the START_TLS input of the TLS_SOCKET function block has occurred. START_TLS was set from TRUE to FALSE during opened TLS socket (ACTIVE input = TRUE). This is the case when:	TLS_*
16#C201	There are too many open sockets in the underlying socket provider.	
16#C202	An operation on a nonblocking socket cannot be completed immediately.	
16#C204	The datagram is too long.	

16#C205	Only one use of an address is normally permitted.	
16#C206	The selected IP address is not valid in this context.	
16#C207	The connection was aborted by the .NET Framework or the underlying socket provider.	
16#C208	The connection was reset by the remote peer.	
16#C210	The application tried to send or receive data, and the Socket is not connected (_SOCKET.ACTIVE == False).	
16#C211	No such host is known. The name is not an official host name or alias.	
16#C212	An unspecified System.Net.Sockets. Socket error has occurred.	
16#C213	The remote host is actively refusing a connection.	
16#C214	An invalid argument was supplied to a System.Net.Sockets.Socket member.	
16#C215	A blocking operation is in progress.	
16#C216	The overlapped operation was aborted due to the closure of the System.Net.Sockets.Socket.	
16#C217	The application has initiated an overlapped operation that cannot be completed immediately.	
16#C218	A blocking System.Net.Sockets.Socket call was canceled.	
16#C219	An attempt was made to access a System.Net.Sockets.Socket in a way that is forbidden by its access permissions.	
16#C21A	An invalid pointer address was detected by the underlying socket provider.	
16#C21B	A System.Net.Sockets.Socket operation was attempted on a non-socket.	
16#C21C	A required address was omitted from an operation on a System.Net.Sockets.Socket.	
16#C21D	An unknown, invalid, or unsupported option or level was used with a System.Net.Sockets.Socket.	
16#C21E	The protocol type is incorrect for this System.Net.Sockets.Socket.	
16#C21F	The protocol is not implemented or has not been configured.	
16#C220	The support for the specified socket type does not exist in this address family.	
16#C221	The address family is not supported by the protocol family.	
16#C222	The protocol family is not implemented or has not been configured.	
16#C223	he address family specified is not supported. This error is returned if the IPv6 address family was specified and the IPv6 stack is not installed on the local machine. This error is returned if the IPv4 address family was specified and the IPv4 stack is not installed on the local machine.	
16#C224	The network is not available.	
16#C225	No route to the remote host exists.	
16#C226	The application tried to set System.Net.Sockets.SocketOptionName. KeepAlive on a connection that has already timed out.	
16#C227	No free buffer space is available for a System.Net.Sockets.Socket operation.	
16#C228	A request to send or receive data was disallowed because the System.Net.Sockets.Socket has already been closed.	
16#C229	The connection attempt timed out, or the connected host has failed to respond.	
16#C22A	The operation failed because the remote host is down.	

16#C22B	There is no network route to the specified host. Could not connect to DEST_IP.	
16#C22C	Too many processes are using the underlying socket provider.	
16#C22D	The network subsystem is unavailable.	
16#C22E	The version of the underlying socket provider is out of range.	
16#C22F	The underlying socket provider has not been initialized.	
16#C230	A graceful shutdown is in progress.	
16#C231	The specified class was not found.	
16#C232	The name of the host could not be resolved. Try again later.	
16#C233	The error is unrecoverable or the requested database cannot be located.	
16#C234	The requested name or IP address was not found on the name server.	

8.2 Data types

```
TYPE
    IPC_ARR_BYTE_0_1459     :   ARRAY   [0..1459]   OF BYTE;

    IPC_UDT_DATA    :   STRUCT              (*Snd/Rcv Data*)
        udiCNT      :   UDINT;              (*number of bytes to be send/
                                            successfully received bytes*)
        arrData     :   IPC_ARR_BYTE_0_1459;(*array for send/rcv data*)
    END_STRUCT;

    IPC_ARR_UDT_DATA_1_20 : ARRAY [1..20] OF IPC_UDT_DATA;

    IPC_UDT_IP_CONNECT  :   STRUCT  (*Connect Data*)
    (*Inputs - extern*)
        xConnect        :   BOOL;   (*rising edge from master to
                                    establish a connection*)
        xIS_SRV         :   BOOL;   (*TRUE - server socket
                                    FALSE - client socket*)
        strDEST_IP      :   STRING; (*depends on xIS_SRV for TCP
                                    TRUE - IP of the client
                                    FALSE - IP of the server*)
        uiDEST_Port     :   UINT;   (*depends on xIS_SRV for TCP
                                    TRUE - Port of the client
                                    FALSE - Port of the server*)
        xStartTLS       :   BOOL;   (* TRUE - start TLS connection *)
        tTimeout        :   TIME;   (*Time for connect timeout*)
        udtConnectInfo  :   CONNECTINFO; (*connection info for TLS_SOCKET*)
        uiBind_Port     :   UINT;   (*Defines the local port number*)
        strBind_IP      :   STRING; (*Defines the local IP address *)
    (*Inputs - intern*)
        xConnected      :   BOOL;   (*connection exists*)
        xActivate       :   BOOL;   (*activate the TCP/UDP_SOCKET*)
    (*Outputs*)
        xActive         :   BOOL;   (*connection is valid*)
        xBusy           :   BOOL;   (*UDP/TCP_SOCKET is busy*)
        xError          :   BOOL;   (*Error of UDP/TCP_SOCKET*)
        wStatus         :   WORD;   (*Status of UDP/TCP_SOCKET *)
        uiUsed_Port     :   UINT;   (*used port of UDP/TCP_Socket*)
        dwHandle        :   DWORD;  (*communication with the
                                    SEND/RECEIVE FB*)
    END_STRUCT;

    IPC_UDT_IP_SEND :   STRUCT  (*Send*)
    (*Inputs - extern*)
        udiDataCNT  :   UDINT;  (*number of bytes to be sent*)
        strDEST_IP  :   STRING; (*DEST_IP for UDP_SEND*)
        uiDEST_Port :   UINT;   (*DEST_PORT for UDP_SEND*)
        xSendReq    :   BOOL;   (*start send external*)
        tTimeout    :   TIME;   (*time for send timeout*)
    (*Inputs - intern*)
        xReq        :   BOOL;   (*start send data internal*)
    (*Outputs*)
        xDone       :   BOOL;   (*send successful*)
        xError      :   BOOL;   (*error of UDP/TCP_SEND*)
        wStatus     :   WORD;   (*status of UDP/TCP_SEND*)
    (*InOut*)
        udtSndData  :   IPC_UDT_DATA;   (*data to be send*)
    END_STRUCT;

    IPC_UDT_IP_RECEIVE  :   STRUCT  (*Receive*)
    (*Inputs - extern*)
        tTimeout        :   TIME;   (*time for receive timeout*)
```

```
(*Outputs*)
    xNDR            :   BOOL;   (*new data received*)
    xError          :   BOOL;   (*error of UDP/TCP_RECEIVE*)
    wStatus         :   WORD;   (*status of UDP/TCP_RECEIVE*)
    strSource_IP    :   STRING; (*IP address of the device from which data
                                have been received*)
    uiSource_Port   :   UINT;   (*Port number of the device from which data
                                have been received*)
(*InOut*)
    udtRcvData      :   IPC_UDT_DATA;   (*received data*)
END_STRUCT;

IPC_UDT_TEST_SOCKET :   STRUCT  (*Test-struct for SOCKET*)
    xError          :   BOOL;   (*force xError = TRUE*)
    xActive         :   BOOL;   (*force xAcvite = FALSE*)
END_STRUCT;

IPC_UDT_TEST_SEND   :   STRUCT  (*Test-struct for SEND*)
    xError          :   BOOL;   (*force error = TRUE*)
    xDone           :   BOOL;   (*force xDone = FALSE*)
END_STRUCT;

IPC_UDT_TEST_RECEIVE    :   STRUCT  (*Test-struct for RECEIVE*)
    xError              :   BOOL;   (*force xError = TRUE*)
    xNDR                :   BOOL;   (*force xNDR = FALSE*)
END_STRUCT;

IPC_UDT_TEST    :   STRUCT  (*Test-struct*)
    udtSOCKET   :   IPC_UDT_TEST_SOCKET;
    udtSEND     :   IPC_UDT_TEST_SEND;
    udtRECEIVE  :   IPC_UDT_TEST_RECEIVE;
END_STRUCT;

IPC_UDT_TCP_SOCKET  :   STRUCT  (*Inputs and outputs of TCP_SOCKET*)
(* Inputs *)
    xActivate       :   BOOL;
    xIS_SRV         :   BOOL;
    strBIND_IP      :   STRING;
    uiBIND_Port     :   UINT;
    strDEST_IP      :   STRING;
    uiDEST_Port     :   UINT;
(* Outputs *)
    dwHandle        :   DWORD;
    xActive         :   BOOL;
    xBusy           :   BOOL;
    xError          :   BOOL;
    wStatus         :   WORD;
    uiUSED_Port     :   UINT;
END_STRUCT;

IPC_UDT_TCP_SEND    :   STRUCT (*Inputs and outputs of TCP_SEND*)
(* Inputs *)
    xReq            :   BOOL;
    udiData_CNT     :   UDINT;
(* Outputs *)
    xDone           :   BOOL;
    xBusy           :   BOOL;
    xError          :   BOOL;
    wStatus         :   WORD;
END_STRUCT;

IPC_UDT_TCP_RECEIVE :   STRUCT  (*Inputs and outputs of TCP_RECEIVE*)
(* Inputs *)
    xEN_R           :   BOOL;
```

```
    udiEXP_Data_CNT :   UDINT;
(* Outputs *)
    xNDR            :   BOOL;
    xError          :   BOOL;
    wStatus         :   WORD;
    strSource_IP    :   STRING;
    uiSource_Port   :   UINT;
    udiData_CNT     :   UDINT;
END_STRUCT;

IPC_UDT_UDP_SOCKET  :   STRUCT (*Inputs and outputs of TCP_SOCKET*)
(* Inputs *)
    xActivate       :   BOOL;
    strBIND_IP      :   STRING;
    uiBIND_Port     :   UINT;
(* Outputs *)
    dwHandle        :   DWORD;
    xActive         :   BOOL;
    xBusy           :   BOOL;
    xError          :   BOOL;
    wStatus         :   WORD;
    uiUSED_Port     :   UINT;
END_STRUCT;

IPC_UDT_UDP_SEND    :   STRUCT  (*Inputs and outputs of TCP_SEND*)
(* Inputs *)
    xReq            :   BOOL;
    strDEST_IP      :   STRING;
    uiDEST_Port     :   UINT;
    udiData_CNT     :   UDINT;
(* Outputs *)
    xDone           :   BOOL;
    xBusy           :   BOOL;
    xError          :   BOOL;
    wStatus         :   WORD;
END_STRUCT;

IPC_UDT_UDP_RECEIVE :   STRUCT  (*Inputs and outputs of TCP_RECEIVE*)
(* Inputs *)
    xEN_R           :   BOOL;
(* Outputs *)
    xNDR            :   BOOL;
    xError          :   BOOL;
    wStatus         :   WORD;
    strSource_IP    :   STRING;
    uiSource_Port   :   UINT;
    udiData_CNT     :   UDINT;
END_STRUCT;

IPC_UDT_TLS_SOCKET  :   STRUCT  (*Inputs and outputs of TLS_SOCKET*)
(* Inputs *)
    xActivate       :   BOOL;
    xIS_SRV         :   BOOL;
    strBIND_IP      :   STRING;
    uiBIND_Port     :   UINT;
    strDEST_IP      :   STRING;
    uiDEST_Port     :   UINT;
    udtConnectInfo  :   CONNECTINFO;
    xStartTLS       :   BOOL;
(* Outputs *)
    dwHandle        :   DWORD;
    xActive         :   BOOL;
    xBusy           :   BOOL;
    xError          :   BOOL;
```

```
        wStatus         :   WORD;
        uiUSED_Port     :   UINT;
    END_STRUCT;

    IPC_UDT_TLS_SEND    :   STRUCT (*Inputs and outputs of TLS_SEND*)
    (* Inputs *)
        xReq            :   BOOL;
        udiData_CNT     :   UDINT;
        xSendSecure     :   BOOL;
    (* Outputs *)
        xDone           :   BOOL;
        xBusy           :   BOOL;
        xError          :   BOOL;
        wStatus         :   WORD;
    END_STRUCT;

    IPC_UDT_TLS_RECEIVE :   STRUCT  (*Inputs and outputs of TLS_RECEIVE*)
    (* Inputs *)
        xEN_R           :   BOOL;
        udiEXP_Data_CNT :   UDINT;
        xReceiveSecure  :   BOOL;
    (* Outputs *)
        xNDR            :   BOOL;
        xError          :   BOOL;
        wStatus         :   WORD;
        strSource_IP    :   STRING;
        uiSource_Port   :   UINT;
        udiData_CNT     :   UDINT;
    END_STRUCT;

    IPC_UDT_DIAG            :   STRUCT  (*Diag struct*)
        iState              :   INT;    (*actual state of send*)
        iRetry              :   INT;    (*Number of retrues*)
        wDiagCode           :   WORD;   (*actual diag code *)
        wAddDiagCode        :   WORD;   (*actual additional diag code*)
        udtIP_TCP_SOCKET    :   IPC_UDT_TCP_SOCKET;
        udtIP_TCP_SEND      :   IPC_UDT_TCP_SEND;
        udtIP_TCP_RECEIVE   :   IPC_UDT_TCP_RECEIVE;
        udtIP_UDP_SOCKET    :   IPC_UDT_UDP_SOCKET;
        udtIP_UDP_SEND      :   IPC_UDT_UDP_SEND;
        udtIP_UDP_RECEIVE   :   IPC_UDT_UDP_RECEIVE;
        udtIP_TLS_SOCKET    :   IPC_UDT_TLS_SOCKET;
        udtIP_TLS_SEND      :   IPC_UDT_TLS_SEND;
        udtIP_TLS_RECEIVE   :   IPC_UDT_TLS_RECEIVE;
    END_STRUCT;

    IPC_UDT_IP_SOCKET   : STRUCT
        xActivate       : BOOL; (* TRUE - activates the IPC_SOCKET *)
        xActive         : BOOL; (* TRUE - IPC_SOCKET is active *)
        xBusy           : BOOL; (* TRUE - locked for other clients *)
        xError          : BOOL; (* TRUE - Error of IPC_SOCKET *)
        xUDP            : BOOL; (* TRUE - for UDP - FALSE for TCP *)
        xTLS            : BOOL; (* TRUE - for TLS IF xUDP = FALSE *)
        xOccupied       : BOOL; (* TRUE - Interface is occupied by a satellite block *)
        xAutoRetry      : BOOL; (* TRUE - Auto retry is executed *)
        iMaxRetry       : INT;  (* Max Retries before error*)
        udtConnect      : IPC_UDT_IP_CONNECT;
        udtSend         : IPC_UDT_IP_SEND;
        udtReceive      : IPC_UDT_IP_RECEIVE;
        udtTest         : IPC_UDT_TEST;
        udtDiag         : IPC_UDT_DIAG;
    END_STRUCT;
END_TYPE
```

9 Support

For technical support please contact your local PHOENIX CONTACT agency

at https://www.phoenixcontact.com

Owner:

PHOENIX CONTACT Electronics GmbH
Business Unit Automation Systems
System Services
Library Services