

Application Note

Data handling blocks for data transfer via Ethernet UDP for XControls and XV-PLCs

03/14 AN27K26G V3.4

Eaton Industries GmbH, Bonn

Author: O. Weiß, A. Stein

All brand and product names are trademarks or registered trademarks of the owner concerned. All rights reserved, including those of the translation. No part of this application note may be reproduced in any form (printed, photocopy, microfilm or any other process) or processed, duplicated or distributed by means of electronic systems without the expressed written permission of Eaton Ind. GmbH, Bonn.

Date: March 2014

Subject to modifications.

Inhaltsverzeichnis

Data handling blocks for data transfer via Ethernet UDP.....	1
1 General Points	3
1.1 Function	3
1.2 Field of application	3
1.3 Hardware requirements.....	3
1.4 Software requirements	3
1.5 ZIP directory structure	3
1.6 New in UDPcomV3.....	3
2 Design.....	4
2.1 Structure.....	4
2.2 Function blocks and parameters	4
2.2.1 UDPcom_TRANSMIT	4
2.2.2 UDPcom_RECEIVE.....	6
2.2.3 UDPcom_DNSCLIENT	7
3 Commisioning	8
3.1 Integration of the ,connectionless' UDP interfacing.....	8
3.2 Function and operation the ,connectionless' UDP interfacing	8
3.2.1 Transmitter <i>UDPcom_TRANSMIT</i>	8
3.2.2 Receiver <i>UDPcom_RECEIVE</i>	8

1 General Points

1.1 Function

Sending and receiving UDP datagrams via Ethernet UDP with all XC/XV PLCs including an Ethernet communication port. Additionally the lib comprises a non blocking DNS client resolver.

1.2 Field of application

Data transfer via UDP datagrams to variable IP addresses controlled by the application.

1.3 Hardware requirements

XVs, XC200s or HPG with Ethernet.

1.4 Software requirements

easySoft CoDeSys (min. version 2.3.9; by using version 2.3.5 some paths are altered, but at least it works also). The UDPcomV3.lib and the SysLibSockets.lib should be placed in: C:\Program Files\Common Files\CAA-Targets\Moeller V2.3.9\Lib_XXX\ or in an equivalent directory. **In your project please set the maximum cycle time (watchdog) to $\geq 200\text{ms}$ (max. application cycle time + additional 200ms).** Please be aware that every target change in CoDeSys clears this value and so the increased cycle time has to be reentered in the according CoDeSys project field again. Please use only one task to access the lib, since it is not 'thread-save'.

1.5 ZIP directory structure

- 2 Docu: contains the documentation of UDPcomV3 (this file).
- 3 Example: contains UDP project examples
- 4 Lib: UDPcomV3.lib to be used in your projects

1.6 New in UDPcomV3

Transmit and Receive are separated into two function blocks, so that multiple instantiation is possible. Serialisation, load limiting of UDP transmissions via a waiting queues and additional error messages were added. So transmissions can be triggered in a quasi parallel manner by the application. Names of function blocks, variable types and array borders were changed to get a more common and linear appearance of the user interface. Particularly the meaning of 'port' and 'bind port' could be mixed up and the usage of the xRcvData-receive signal has been changed, which might lead to confusion when migrating from legacy projects. Thus, you have exactly to know what is meant and so a pure 'flat' migration of legacy projects is not suggested. Therefore V3 is intended to be used in new UDP-projects. Additionally the lib comprises a non blocking 'DNS client resolver function block'.

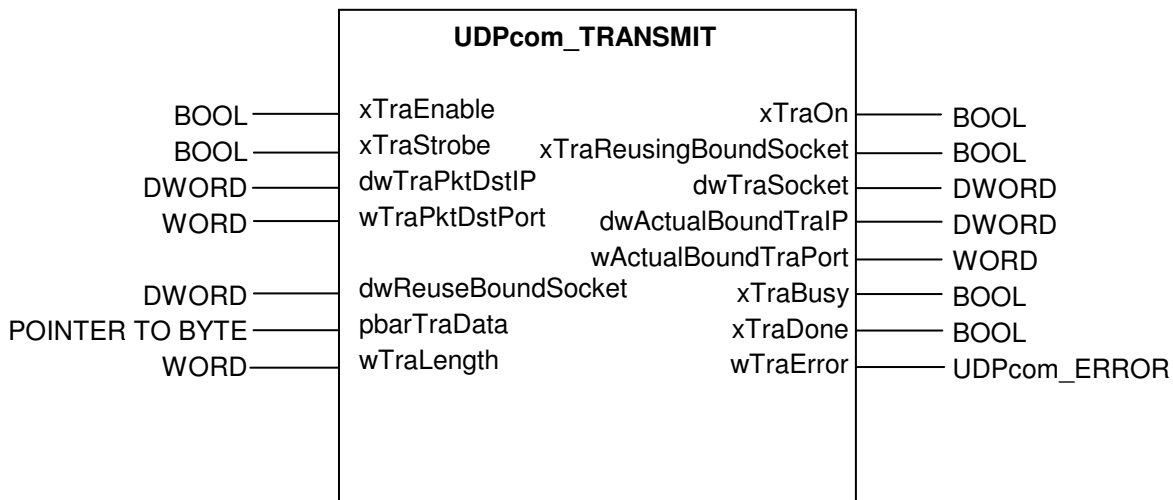
2 Design

2.1 Structure

The function blocks *UDPcom_TRANSMIT* and *UDPcom_RECEIVE* are loaded into the application via the library file *UDPcomV3.lib* and called up from there. The example *UDPcomV3_StandardExample.pro* demonstrates the call of both function block types. The required global variables are automatically loaded into the global variable list *Global_UDPcom*.

2.2 Function blocks and parameters

2.2.1 UDPcom_TRANSMIT



VAR_INPUT	
xTraEnable	TRUE = provides approval for the transmission jobs via this function block FALSE = closes the transmission socket, if existent, of this function block
xTraStrobe	A rising edge triggers a transmission job, which might take more than one cycle steps. Resetting xTraStrobe back to FALSE during an active transmission (xTraBusy=TRUE) leads to a retreat from the transmission with the function block's socket left open (until xTraEnable is set back to FALSE).
dwTraPktDstIP	Destination IP address, in 'big endianness' (e.g. 192.168.119.60 = 16#C0A8773C)
wTraPktDstPort	Destination port number of the datagram to be sent (default value is 10010)
dwReuseBound Socket	0: This function block is supposed to use an own (implicitly bound) transmit socket. >0: This function block (re)uses this already opened receiver socket as transmit socket, with the intention to share the same port and interface for sending and receiving UDP datagrams (this might make sense for e.g. client server applications which identify the service requests and replies by means of the destination and source port).
pbarTraData	Pointer to a data transmit buffer, e.g. ARRAY[1..UDPcom_TRALENGTHMAX]
wTraLength	number of data bytes to be sent (max. UDPcom_TRALENGTHMAX resp. 1450 byte)
xTraSockOptReuse Addr	FALSE = default case: <i>Not using</i> Socket-Option SOCKET_SO_REUSEADDR. TRUE = using Socket-Option SOCKET_SO_REUSEADDR (not recommended).

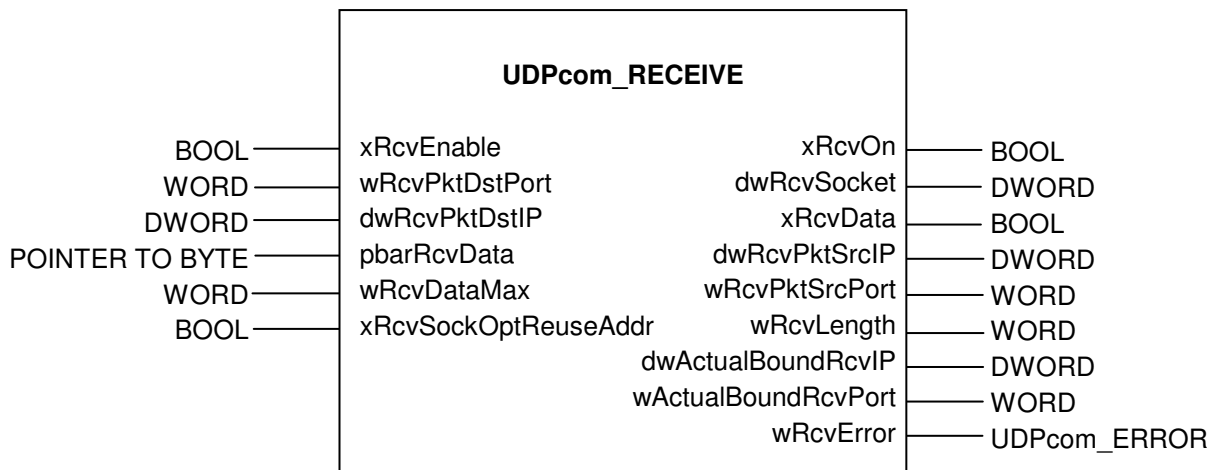
<i>VAR_OUTPUT</i>	
xTraOn	Indicates, whether this function block owns a valid transmit socket at the moment.
xTraReusingBoundSocket	Indicates, whether this function block (re)uses an already opened receiver socket as transmit socket to share a port and the interface with an already bound receiver function block e.g. for sending and receiving UDP client server datagrams (acknowledge of xTraReuseBoundSocket).
dwTraSocket	Transmit socket handle (valid only when xTraOn = TRUE)
wActualBoundTraIP	IP of the interface this function block is currently bound to. 0 means no explicit binding to a dedicated interface, thus the standard IP/interface is used.
wActualBoundTraPort	Port to which this function block is currently bound to. 0 means no explicit binding to a dedicated port, thus an arbitrary free port is used.
xTraBusy (= xTraDone invertiert)	TRUE = function block is busy. FALSE after falling edge and UDPcom_ERROR = UDPcom_OK means: The UDP datagram was delivered to the underlying transmit socket (but not necessarily received / without receipt).
wTraError	Function block error code according UDPcom_ERROR (enum)

UDPcom_TRANSMIT function blocks each own a (non directly bound) socket (dwReuseBoundSocket:=0) or alternatively no own socket (dwReuseBound-Socket := FB_Receive.dwRcvSocket).

The hereafter described UDPcom_RECEIVE function blocks are always bound according FB's user parameters (wRcvPktDstPort and dwRcvPktDstIP) to a dedicated port and interface.

If you wish to transmit via an explicitly bound socket (e.g. client server model), this is only possible by binding the UDPcom_TRANSMIT function block to an already explicitly bound FB_Receive.dwRcvSocket function block and thus to share the receiver blocks socket.

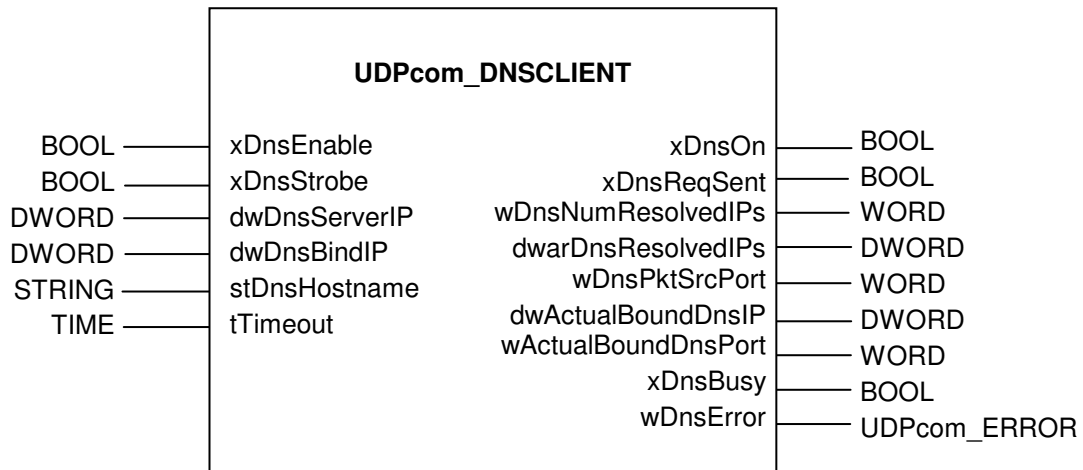
2.2.2 UDPcom_RECEIVE



VAR_INPUT	
xRcvEnable	TRUE = provides admission for receive function of this function block FALSE = closes the receive socket of this function block
wRcvPktDstPort	Sets the receive function block filter in a manner, that only datagrams with fitting destination port numbers (=wRcvPktDstPort) are received by this function block, independently of the source port value in the received datagram.
dwRcvPktDstIP	Defines the binding of the receive socket of this receiver function block to a specific interface. Setting a value wRcvPktDstIP other than 0 only makes sense, if the corresponding IP interface is existent; meaning: using other values than 0 or the current standard IP value only makes sense if the PLC supports multiple IP interfaces and these interfaces are also defined specifically in the PLC. Please leave this parameter set to 0 for e.g.: EC4P-222, XV10x and XC201.
<small>Rcv = Receive-F.Block Pkt = Packet/Datagram Dst = Destination-Field Src = Source-Field</small>	
pbarRcvData	Pointer to a data receive buffer, e.g. ARRAY [1.. UDPcom_RCVLENGTHMAX]
wRcvDataMax	Max. number of data bytes to be received (max. UDPcom_RCVLENGTHMAX respectively 1500 Byte, default value is 0). wRcvDataMax = 0 generates the receive socket only.
xRcvSockOptReuseAddr	FALSE = default case: <i>Not using</i> Socket-Option SOCKET_SO_REUSEADDR. TRUE = Using Socket-Option SOCKET_SO_REUSEADDR (not recommended).
VAR_OUTPUT	
xRcvOn	Indicates, whether this function block currently owns a valid receive socket
dwRcvSocket	Receive socket handle (valid only when xRcvOn = TRUE)
xRcvData	TRUE (level sensitive) = at least one datagram was received
dwRcvPktSrcIP	Source IP address, in big endianness (e.g. 192.168.119.60 = 16#C0A8773C)
wRcvPktSrcPort	Source port value of the received datagram. This has not necessarily the same value as the received destination port, which is equivalent to wRcvPktDstPort. It is recommended not to refer to wRcvPktSrcPort (because of IP masquerading of routers).
wRcvLength	Number of data bytes of the received datagram.
dwActualBoundTraIP	IP of the interface this function block is currently bound to. 0 means no explicit binding to a dedicated interface, thus the standard IP/interface is used.
wActualBoundTraPort	Port to which this function block is currently bound to UDP receiver sockets are always bound to a dedicated port.
wRcvError	Function block error code according UDPcom_ERROR (enum)

2.2.3 UDPcom_DNSCLIENT

The function block *UDPcom_DNSCLIENT* is loaded into the application via the library file *UDPcomV3.lib* and called up from there. The example *UDPcomV3_DnsResolverExample.pro* demonstrates the call of this non blocking function block type.



VAR_INPUT	
xDnsEnable	TRUE = provides admission for DNS function of this function block FALSE = closes the UDP socket of this function block
xDnsStrobe	A rising edge triggers a DNS request, which might take more than one cycle steps. Resetting xDnsStrobe back to FALSE during an active request (xDnsBusy=TRUE) leads to a retreat from the resolve attempt with a forced timeout error.
dwDnsServerIP	DNS server's IP address, in 'big endianness' (e.g. 192.168.119.001 = 16#C0A87701)
dwDnsBindIP	Defines the binding of the UDP socket of this DNS resolver function block to a specific interface. Setting a value wDnsBindIP other than 0 only makes sense, if the corresponding IP interface is existent; meaning: using other values than 0 or the current standard IP value only makes sense if the PLC supports multiple IP interfaces and these interfaces are also defined specifically in the PLC. Please leave this parameter set to 0 for e.g.: EC4P-222, XV10x and XC201.
stDnsHostname	Hostname to be resolved like e.g.: 'www.moeller.net'
tTimeout	Maximum accepted DNS-Server reply time (timeout).
VAR_OUTPUT	
xDnsOn	Indicates, whether this function block owns a valid UDP socket at the moment.
xDnsReqSent	State: A 'DNS-Client-Resolver-Request' was sent to the server.
wDnsNumResolvedIPs	If wDnsError = UDPcom_OK: number of resolved IP-addresses received from the DNS server and stored in dwDnsResolvedIPs.
dwarDnsResolvedIPs	If wDnsError = UDPcom_OK: resolved IP addresses, beginning with index 1
dwActualBoundDnsIP	IP of the interface this function block is currently bound to. 0 means no explicit binding to a dedicated interface, thus the standard IP/interface is used.
wActualBoundDnsPort	Port to which this function block is currently bound to UDP receiver sockets are always bound to a dedicated port. 53 defines the DNS standard port.
xDnsBusy	TRUE = function block is busy, FALSE after falling edge and UDPcom_ERROR = UDPcom_OK means: DNS service reply received.
wDnsError	Function block error code according UDPcom_ERROR (enum)

3 Commisioning

3.1 Integration of the ,connectionless' UDP interfacing

The example file *EC_UDPcomV3_StandardExample.pro* should be loaded. The function block *UDPcom_TRANSMIT* is able to send data to several IP addresses via one instance, but anyway, it is recommended to use an own transmission socket for each receiver IP as long as this doesn't consume too much sockets (< 8 sockets). This avoids creating and closing the socket each time you want to connect to another IP. Each function block instance creates its own socket. If receiving data via several ports is required, there should be created one own instance of the FB *UDPcom_RECEIVE* for each port. If you wish to avoid parametric programming of the function block, you can use *UDPcom_StandardExample* as an example program. This contains the function block calls with the more important parameters. The *UDPcomV3.lib* requires the *LibSockets.lib*, *SysLibCallback.lib* and *Standard.lib* libraries, whereby those will also be imported with the selection of *UDPcomV3.lib* automatically, as long as these libraries are provided.

notice: Only the function blocks *UDPcom_RECEIVE*, *UDPcom_TRANSMIT*, *UDPcom_DNSCLIENT* and the function *UDPcom_MakeIP* should be called by the application.

3.2 Function and operation the ,connectionless' UDP interfacing

3.2.1 Transmitter *UDPcom_TRANSMIT*

xTraEnable opens the transmit socket. The socket remains open until *xTraEnable* is reset or until the PLC is stopped. The Socket handle is displayed with the output parameter *dwTraSocket*. *xTraOn* acknowledges the opened socket and the readiness to transmit. A rising edge on *xTraStrobe* places a transmit job on the IP *dwTraPktDstIP* via the *wTraPktSrcPort* port (source port) to the destination port *wTraPktDstPort*. It is recommended to leave *wTraPktSrcPort* untouched to its default value zero. In the process, the number of bytes *wTraLength* is transmitted from the *barTraData* buffer. *xTraBusy* = *FALSE* signals that the send job has been placed onto the transmitter socket. Errors are displayed on the *wTraError* output.

If the transmit job has to be placed via a fixed port, it can be defined with the input parameter *wTraPktSrcPort*. The required port number has to be entered here to bind it to the transmit socket. If a '0' is entered, which is recommended, the system selects an quasi random transmit port and thus the socket is not bound to a defined transmit socket.

Caution: UDP is connectionless, i.e. the receiver does not automatically acknowledge receipt of a telegram (-> perhaps an acknowledgement telegram might be sent back via the acknowledgement telegram).

The transmitter sends its telegrams into the receive queue of the receiver. The size of the receive queue depends on the hardware. The receiver reads the telegrams out of its receive queue in the same order as they were received. To avoid reading 'old' telegrams into the application, don't send more often telegrams as the receiver reads them -> handshake !

Because of this features an event driven communication concept should be avoided using UDP. It would be more recommended to use UDP within a cyclic or handshake based communication model.

The transmitter has broadcast capability (IP address 255.255.255.255).

3.2.2 Receiver *UDPcom_RECEIVE*

xRcvEnable opens the receive socket (*dwRcvSocket*). The socket remains open until *xRcvEnable* is reset or until the PLC is stopped. *xRcvOn* acknowledges the open socket and readiness to receive. *wRcvPktDstPort* determines the port via which the telegrams are received. The port number must correspond with the received destination port number. It is important to ensure that no system ports are used here. *xRcvData* is set as long as data are in the receive buffer. *dwRcvPktSrcIP* indicates the sender and *wRcvLength* indicate how many data bytes have been received and copied in the buffer referred by *pbarRcvData*. *wRcvError* signals errors e.g. when opening or closing the socket and telegram length faults.

Hint: the contents of the global UDP variables in *Global_UDPcom* might be helpful for runtime error analysis.