

# Anwendungshinweis

## Hantierungsbausteine zum Datentransfer über Ethernet UDP für XControl- und XV- Steuerungen

---

**03/14 AN27K22G V3.4**

© Eaton Industries GmbH, Bonn

Autor: O. Weiß, A. Stein

Alle Marken- und Produktnamen sind Warenzeichen oder eingetragene Warenzeichen der jeweiligen Titelfalter.

Alle Rechte, auch die der Übersetzung, vorbehalten. Kein Teil dieses Anwendungshinweises darf in irgendeiner Form (Druck, Fotokopie, Mikrofilm oder einem anderen Verfahren) ohne schriftliche Zustimmung der Firma Eaton Industries GmbH, Bonn, reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Stand: März 2014

Änderungen vorbehalten.

## Inhaltsverzeichnis

Hantierungsbausteine zum Datentransfer über Ethernet UDP .....	1
1 Allgemeines .....	3
1.1 Funktion .....	3
1.2 Einsatzbereich.....	3
1.3 Hardwarevoraussetzungen .....	3
1.4 Softwarevoraussetzung.....	3
1.5 ZIP-Verzeichnisstruktur .....	3
1.6 Neu in UDPcomV3 .....	3
2 Aufbau .....	4
2.1 Struktur.....	4
2.2 Funktionsbausteine und Parameter .....	4
2.2.1 UDPcom_TRANSMIT .....	4
2.2.2 UDPcom_RECEIVE.....	6
2.2.3 UDPcom_DNSCLIENT .....	7
3 Inbetriebnahme.....	8
3.1 Einbindung der „verbindungslosen“ UDP Ankopplungen.....	8
3.2 Arbeitsweise und Bedienung der „verbindungslosen“ UDP Ankopplung.....	8
3.2.1 Sender <i>UDPcom_TRANSMIT -Funktionsbaustein</i> .....	8
3.2.2 Empfänger <i>UDPcom_RECEIVE -Funktionsbaustein</i> .....	8

## 1 Allgemeines

### 1.1 Funktion

Senden und Empfang von UDP-Datagrammen über Ethernet-UDP auf allen XC/XV-Steuerungen mit Ethernet Kommunikations-Schnittstelle. Zusätzlich ist ein nichtblockierender UDP-DNS-Client (IP-Resolver) enthalten.

### 1.2 Einsatzbereich

Datentransfer an variable IP-Adressen (auch zum oder vom PC), aus der Applikation heraus gesteuert.

### 1.3 Hardwarevoraussetzungen

XVs, XC200er, HPG mit Ethernet.

### 1.4 Softwarevoraussetzung

easySoft CoDeSys (ab Version 2.3.9; bei Version 2.3.5 sind die Pfade etwas anders, geht aber prinzipiell auch). Die UDPcomV3.lib und die SysLibSockets.lib sollten in C:\Programme\Gemeinsame Dateien\CAA-Targets\Moeller V2.3.9\Lib\_xxx\ oder in einem entsprechenden Verzeichnis liegen. Die maximale Projekt-Zykluszeit (**Task-Zyklus-Watchdog**) **bitte auf mindestens 200ms setzen, (max. Applikationszykluszeit + zusätzliche 200ms)**. Jeder Target-Wechsel in der CoDeSys überschreibt diesen Wert, so daß er dann erneut eingegeben werden muß. Alle UDPcom-Funktionsblöcke müssen in derselben „Task“ laufen.

### 1.5 ZIP-Verzeichnisstruktur

Docu: Enthält die Dokumentation von UDPcomV3 (diese Datei).  
Example: Enthält UDP-Beispielprojekte  
Lib: UDPcomV3.lib enthält die UDP-Funktionalität als Bibliothek.

### 1.6 Neu in UDPcomV3

„Transmit“ und „Receive“ wurden in separate Funktionsbausteine getrennt, so daß nunmehr mehrere Instanzen möglich sind. Hinzugekommen sind eine Serialisierung und Lastlimitierung der UDP-Sendeaufträge via Warteschlange und zusätzliche Fehlermeldungen. Dadurch können die Sendeaufträge voneinander unabhängig, quasiparallel gesendet werden. Es wurden die Namen der Funktionsbausteine, Variablentypen und Array-Grenzen abgeändert, um eine einheitlichere und linearere Linie zu erhalten. Besonders bzgl. der Verwendung von Port und Bind-Port und des xRcvData-Empfangssignals besteht bei der Migration älterer UDP-Projekte auf diese Version Verwechslungsgefahr. Hier muß bei der Migration darauf geachtet werden, was jeweils gemeint ist. Eine rein formelle Migration wird also nicht empfohlen. Die Version V3 ist daher hauptsächlich für die Verwendung in neuen UDP-Projekten gedacht. Desweiteren ist ab V3.1 ein nichtblockierender UDP-DNS-Client-Resolver-Funktionsbaustein enthalten.

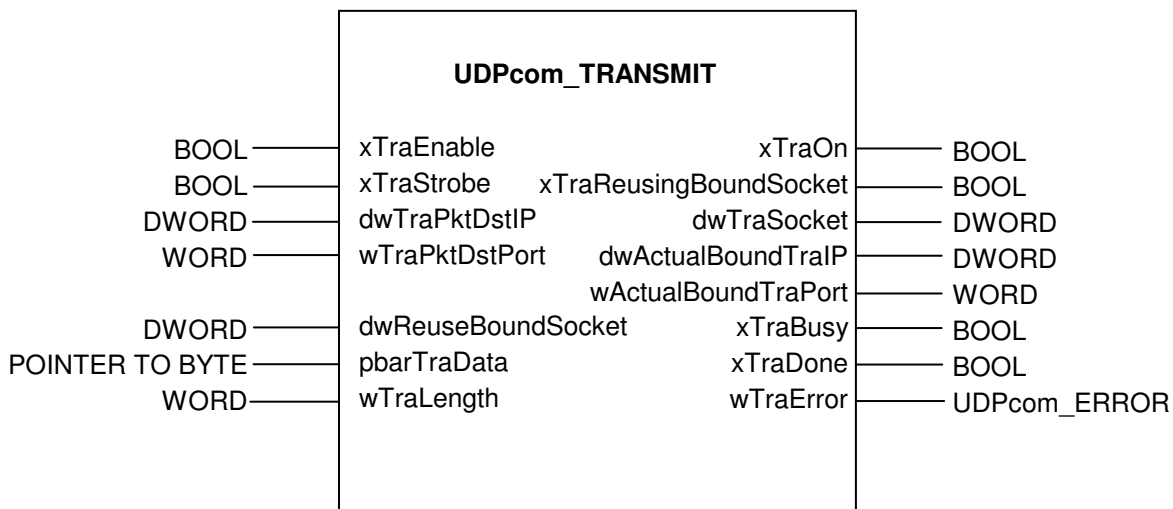
## 2 Aufbau

### 2.1 Struktur

Die Funktionsbausteine *UDPcom\_TRANSMIT* und *UDPcom\_RECEIVE* werden über die Bibliothek *UDPcomV3.lib* in die Applikation geladen und von dort aus aufgerufen. Das Beispielprojekt *UDPcomV3\_StandardExample.pro* zeigt beispielhaft den Aufruf der beiden Funktionsbausteine. Die benötigten globalen Variablen werden automatisch in die globale Variablenliste *Global\_UDPcom* hinein geladen.

### 2.2 Funktionsbausteine und Parameter

#### 2.2.1 UDPcom\_TRANSMIT



VAR_INPUT	
xTraEnable	TRUE = Freigabe für die Sendeaufträge dieses Funktionsblocks erteilen FALSE = den Sendesockel dieses Funktionsblocks, falls vorhanden, schließen
xTraStrobe	Ein Sendeauftrag für ein Datagramm wird durch eine steigende Flanke gestartet und kann mehrere Schritte dauern. Setzt man xTraStrobe wieder auf FALSE während xTraBusy noch TRUE ist, so wird der Sendevorgang abgebrochen und der Sendesockel bleibt geöffnet (bis xTraEnable wieder auf FALSE gesetzt wird).
dwTraPktDstIP	Ziel-IP-Adresse, in "Big Endianness"-Notation (z.B. 192.168.119.60 = 16#C0A8773C)
wTraPktDstPort	Ziel-Port-Nummer des zu sendenden Datagramms (Default-Wert ist 10010)
dwReuseBoundSocket	0: Dieser Funktionsbaustein soll einen eigenen Sendesockel benutzen. >0: Dieser FB soll diesen bereits geöffneten UDP-Empfangssockel als Sendesockel benutzen, um dessen Port & Interface mitnutzen zu können (z.B. für Client-Server).
pbarTraData	Zeiger auf den Daten-Sendepuffer, z.B. auf ARRAY[1..UDPcom_TRALENGTHMAX].
wTraLength	Anzahl der zu sendenden Daten-Bytes (max. UDPcom_TRALENGTHMAX 1450 Byte) Empfohlen werden jedoch max. 460 Nutz-Bytes lange Sende-Datagramme. Setzt man wTraLength auf 0, so wird kein Sendevorgang gestartet und der Sendesockel bleibt geöffnet (bis xTraEnable wieder auf FALSE gesetzt wird).
xTraSockOptReuseAddr	FALSE = Normalfall: Die Socket-Option SOCKET_SO_REUSEADDR nicht benutzen. TRUE = die Socket-Option SOCKET_SO_REUSEADDR für den Sendesockel dieses Funktionsblocks während dessen Generierung setzen (wird nicht empfohlen).

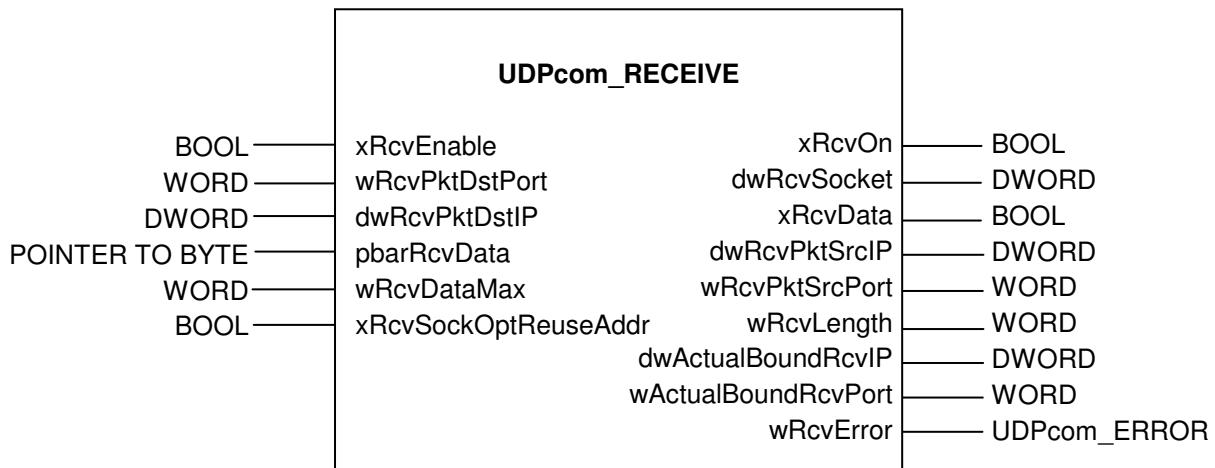
<i>VAR_OUTPUT</i>	
xTraOn	Zeigt an, ob dieser Funktionsblock z.Zt. einen eigenen erzeugten Sendesocket besitzt.
xTraReusingBoundSocket	Zeigt an, ob dieser Funktionsblock einen bereits geöffneten Empfangssocket als Sendesocket mitbenutzt, um z.B. bei UDP-Client-Server-Anwendungen über dessen Port (mit)kommunizieren zu können (Bestätigung von xTraReuseBoundSocket).
dwTraSocket	Transmit Socket Handle (dieser ist nur bei xTraOn = TRUE gültig)
wActualBoundTraIP	IP des Interface an welches der Funktionsblock momentan gebunden ist. 0 bedeutet keine explizite Bindung an ein Interface; d.h. die Standard-IP wird verwendet.
wActualBoundTraPort	Port an welchem der Funktionsblock momentan gebunden ist. 0 bedeutet keine explizite Bindung; d.h. es wird ein freier, quasi-zufälliger Port verwendet.
xTraBusy (= xTraDone invertiert)	TRUE = der Funktionsblock ist noch beschäftigt. FALSE in Kombination mit wTraError = UDPcom_OK bedeutet: Das UDP-Datagramm wurde (nur) dem Sendesocket übergeben (dies ist jedoch keine echte Sende- oder gar Empfangsbestätigung).
wTraError	Funktionsblock-Fehlercode, siehe Datentypen in UDPcom_ERROR (enum)

UDPcom\_TRANSMIT-Funktionsbausteine besitzen jeweils einen eigenen indirekt gebundenen Socket (dwReuseBoundSocket:=0) oder alternativ gar keinen eigenen Socket (dwReuseBound-Socket := FB\_Receive.dwRcvSocket).

Die im folgenden beschriebenen UDPcom\_RECEIVE-Funktionsbausteine sind hingegen immer gemäß der Benutzerparameter (wRcvPktDstPort und dwRcvPktDstIP) an einen Port und an ein IP-Interface gebunden.

Will man über einen explizit gebundenen Socket senden (Client-Server), so geht das nur indem man den UDPcom\_TRANSMIT-Funktionsbaustein mittels z.B. dwReuseBound-Socket := FB\_Receive.dwRcvSocket an einen passend gebundenen UDPcom\_RECEIVE-Funktionsbaustein bindet.

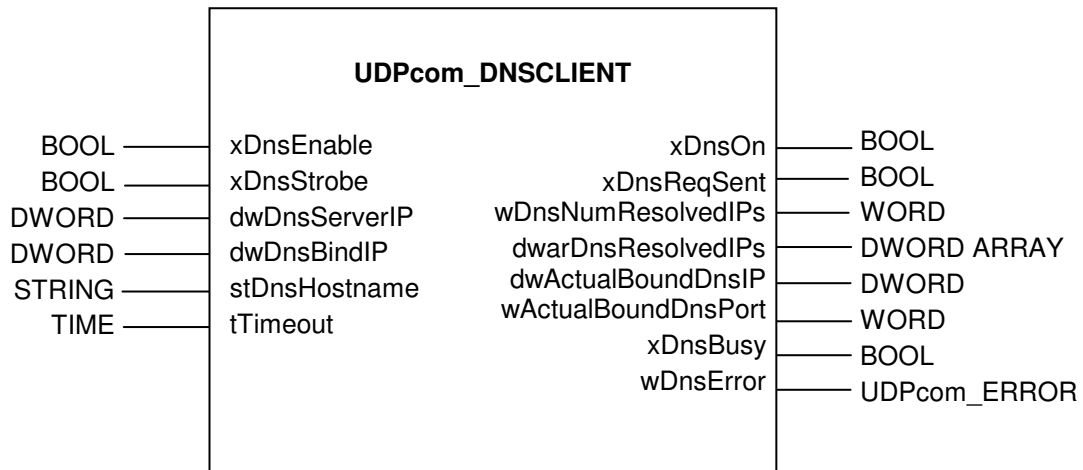
### 2.2.2 UDPcom\_RECEIVE



VAR_INPUT	
xRcvEnable	TRUE = Freigabe für den Empfang über diesen Funktionsblock erteilen FALSE = den Empfangssocket dieses Funktionsblocks schließen
wRcvPktDstPort	Setzt den Funktionsblock-Empfangsfilter so, daß nur Datagramme mit gleich lautendem Ziel-Port (=wRcvPktDstPort) von diesem Funktionsbaustein empfangen werden, unabhängig vom Wert des Quell-Ports im empfangenen Datagramm. Der fragliche Port darf jedoch nicht bereits von einem anderen Funktionsbaustein belegt (gebunden) sein, ansonsten erhält man einen Bindungs-Fehler zurück.
dwRcvPktDstIP	Definiert die Bindung des Empfangssockels dieses Funktionsblocks an ein spezifisches Interface. Werte von wRcvPktDstIP ungleich 0 sind nur zulässig, wenn das entsprechende Interface auch existiert; d.h. andere Werte als 0 oder die aktuelle Standard-IP machen hier nur Sinn, wenn die Steuerung mehrere IP-Interfaces unterstützt und diese auch definiert sind. Dieser Parameter sollte 0 bleiben z.B. für die PLCs: EC4P-222, XV10x und XC201.
pbarRcvData	Zeiger auf den Daten-Empfangspuffer, z.B. ARRAY[1.. UDPcom_RCVLENGTHMAX]
wRcvDataMax	Anzahl der maximal zu empfangenen Daten-Bytes (max.UDPcom_RCVLENGTHMAX bzw. 1500 Byte, Default = 0). wRcvDataMax = 0 generiert nur den Empfangssocket.
xRcvSockOptReuseAddr	FALSE = Normalfall: die Socket-Option SOCKET_SO_REUSEADDR nicht benutzen. TRUE = die Socket-Option SOCKET_SO_REUSEADDR für den eigenen Empfangssocket während dessen Generierung setzen (wird nicht empfohlen).
VAR_OUTPUT	
xRcvOn	Zeigt an, ob dieser Funktionsblock momentan einen erzeugten Empfangssocket besitzt.
dwRcvSocket	Receive Socket Handle (Dieser ist nur bei xRcvOn = TRUE gültig)
xRcvData	TRUE = es wurde mindestens ein Datagramm empfangen
dwRcvPktSrcIP	Quell-IP-Adresse, in Big Endianness (z.B. 192.168.119.60 = 16#C0A8773C)
wRcvPktSrcPort	Quell-Port des empfangenen Datagramms. Dieser ist nicht zwangsläufig identisch mit dem empfangenen Ziel-Port, welcher in wRcvPktDstPort steht.
wRcvLength	Anzahl der empfangenen Datagramm-Daten-Bytes
dwActualBoundTraIP	IP des Interface an welches der Socket des Funktionsblocks momentan gebunden ist.
wActualBoundTraPort	Port an welchem der Socket des Funktionsblocks momentan gebunden ist. Empfangsfunktionsblöcke bzw. deren UDP-Empfangssocket werden immer gebunden.
wRcvError	Funktionsblock-Fehlercode, siehe Datentypen in UDPcom_ERROR (enum)

### 2.2.3 UDPcom\_DNSCLIENT

Der Funktionsbaustein *UDPcom\_DNSCLIENT* wird über die Bibliothek *UDPcomV3.lib* in die Applikation importiert und von dort aus aufgerufen. Das Beispielprojekt *UDPcomV3\_DnsResolverExample.pro* zeigt beispielhaft den Aufruf dieses nichtblockierenden „DNS-Client“-Funktionsbausteins.



VAR_INPUT	
<i>xDnsEnable</i>	TRUE = Freigabe für diesen DNS-Client-Funktionsblock erteilen FALSE = den UDP-Sockel dieses Funktionsblocks schließen
<i>xDnsStrobe</i>	Ein „DNS-Resolver“-Auftrag wird durch eine steigende Flanke gestartet und kann mehrere Zyklen dauern. Setzt man <i>xDnsStrobe</i> wieder auf FALSE während <i>xDnsBusy</i> noch TRUE ist, so wird ein vorzeitiges Timeout erzwungen.
<i>dwDnsServerIP</i>	DNS-Server-IP-Adresse, in Big Endianness (z.B. 192.168.119.001 = 16#C0A87701)
<i>dwDnsBindIP</i>	Definiert die Bindung des UDP-Sockels dieses Funktionsblocks an ein spezifisches Interface. Aus technischer Sicht definiert <i>dwDnsBindIP</i> die Source-IP der abgesendeten DNS-Anfrage und auch die Destination-IP der zugehörigen zu empfangenden DNS-Antwort. Werte von <i>wTraPktDstIP</i> ungleich 0 sind nur zulässig, wenn das entsprechende Interface auch existiert; d.h. andere Werte als 0 oder die aktuelle Standard-IP machen hier nur Sinn, wenn die Steuerung mehrere IP-Interfaces unterstützt und diese auch definiert sind. Dieser Parameter sollte 0 bleiben z.B. für die PLCs: XV10x, XC201 und EC4P-222.
<i>stDnsHostname</i>	In eine IP-Adresse aufzulösender Hostname wie z.B.: 'www.moeller.net'
<i>tTimeout</i>	Maximal geduldete Wartezeit (Timeout) für die Antwort des DNS-Servers.
VAR_OUTPUT	
<i>xDnsOn</i>	Zeigt an, ob dieser Funktionsblock momentan einen erzeugten UDP-Sockel besitzt.
<i>xDnsReqSent</i>	Status: Ein „DNS-Client-Resolver-Request“ wurde an den Server abgesendet.
<i>wDnsNumResolvedIPs</i>	Falls <i>wDnsError</i> = UDPcom_OK: Anzahl der erhaltenen aufgelösten IP-Adressen in <i>dwDnsResolvedIPs</i> .
<i>dwarDnsResolvedIPs</i>	Falls <i>wDnsError</i> = UDPcom_OK: aufgelöste IP-Adressen, angefangen bei Index 1
<i>dwActualBoundDnsIP</i>	IP des Interface an welches der Sockel des Funktionsblocks momentan gebunden ist.
<i>wActualBoundDnsPort</i>	Port an welchem der Sockel des Funktionsblocks momentan gebunden ist.
<i>xDnsBusy</i>	TRUE = der Funktionsblock ist noch beschäftigt. FALSE in Kombination mit <i>wDnsError</i> = UDPcom_OK bedeutet: Es wurde eine DNS-Server-Antwort empfangen.
<i>wDnsError</i>	Funktionsblock-Fehlercode, siehe Datentypen in UDPcom_ERROR (enum)

## 3 Inbetriebnahme

### 3.1 Einbindung der „verbindungslosen“ UDP Ankopplungen

Das Beispielprojekt *UDPcomV3\_StandardExample.pro* sollte sowohl bei dem Sender als auch beim Empfänger geladen werden. Der Funktionsbaustein *UDPcom\_TRANSMIT* kann auch über eine Instanz nacheinander an mehrere IP-Adressen senden. Trotzdem wird empfohlen, für jede Ziel-IP einen eigenen Funktionsblock zu nutzen, solange dieses nicht zu viele Sockel benötigt (<<8 Sockel). Dieses vermeidet ein unnötiges und auch zykluszeitraubendes Öffnen und Schließen von Sendesockeln. Will man über mehrere Ports empfangen, so muß für jeden Port eine Instanz des FBs *UDPcom\_RECEIVE* angelegt werden. Will man sich die eigene Parametrierung des FBs ersparen, so kann man auch das Beispielprogramm *UDPcomV3\_StandardExample.pro* aufrufen und abwandeln. Dieses enthält exemplarisch jeweils einen FB-Aufruf mit den nötigsten Parametern. Die *UDPcomV3.lib* benötigt die Bibliotheken *SysLibSockets.lib*, *SysLibCallback.lib* und *Standard.lib*, wobei diese Libs, falls vorhanden, automatisch mit der *UDPcomV3.lib* geladen werden.

Beachte: Nur die Funktionsblöcke *UDPcom\_RECEIVE*, *UDPcom\_TRANSMIT*, *UDPcom\_DNSCLIENT* und die Funktion *UDPcom\_MakeIP* sollten vom Anwender verwendet werden.

### 3.2 Arbeitsweise und Bedienung der „verbindungslosen“ UDP Ankopplung

#### 3.2.1 Sender *UDPcom\_TRANSMIT* -Funktionsbaustein

*xTraEnable* öffnet den Transmit Socket. Der Socket bleibt geöffnet bis *xTraEnable* wieder zurückgesetzt wird, oder bis die Steuerung angehalten wird. Der Socket-Handle kann bei Bedarf einem entsprechenden FB-Ausgang entnommen werden. *xTraOn* quittiert den geöffneten Socket und die Bereitschaft zum Senden. *UDPcom\_ActiveSocketCounter* gibt Auskunft über die Gesamtzahl der geöffneten Sockel. Eine steigende Flanke an *xTraStrobe* setzt einen Sendeauftrag an die IP *dwTraPktDstIP* mit dem Zielport *wTraPktDstPort* über den Quell-Port *wTraPktSrcPort* (optional) ab. Dabei wird die Anzahl *wTraLength* an Byte aus dem Buffer *barTraData* gesendet. *xTraBusy* = FALSE signalisiert, daß der Sendeauftrag an den Sockel abgesetzt wurde, nicht jedoch daß das Datagramm auch wirklich empfangen wurde (UDP sendet verbindungslos). Fehler werden am Ausgang *wTraError* angezeigt. Soll der Sendeauftrag über einen festen Port abgesetzt werden, so kann dieser mit dem Parameter *wTraPktSrcPort* definiert werden. Hierfür ist dieser Parameter mit der entsprechenden Port-Nummer zu belegen, um den Port an den Sendesockel zu binden. Bei dem Wert '0' (empfohlen) auf diesem Eingang geschieht keine Bindung und das System sucht sich einen freien Port zum senden.

Achtung: UDP ist verbindungslos, d.h. der Empfänger quittiert den Erhalt eines Datagrammes nicht automatisch (-> evtl. ist ein Quittungstelegramm über die Applikation zurückzusenden).

Ebenso überträgt der Sender seine Datagramme in die Empfangsqueue des Empfängers. Die Größe der Empfangsqueue ist hardware-abhängig. Der Empfänger liest die Datagramme aus seiner Empfangsqueue in der Reihenfolge in der sie empfangen wurden. Um zu gewährleisten, daß keine „alten“ Datagramme in die Applikation gelesen werden, darf nicht häufiger geschrieben als gelesen werden. Wegen dieser Einschränkungen sollte eine ereignisbasierte UDP-Kommunikation eher vermieden werden und stattdessen ein zyklisches oder handshake-basiertes Kommunikationsmodell auf UDP aufgesetzt werden.

Der Sender ist broadcast-fähig (IP-Adresse 255.255.255.255).

#### 3.2.2 Empfänger *UDPcom\_RECEIVE* -Funktionsbaustein

*xRcvEnable* öffnet einen Empfangssockel (*dwRcvSocket*). Der Sockel bleibt geöffnet bis *xRcvEnable* wieder zurückgesetzt wird, oder bis die Steuerung angehalten wird. *xRcvOn* quittiert den geöffneten FB-Sockel und die Bereitschaft zum Empfangen. *wRcvPktDstPort* bestimmt den Port über den die Datagramme für einen gewissen Funktionsbaustein empfangen werden. Die Port-Nummer muß mit der des empfangenen Ziel-Ports, aber nicht unbedingt mit dem Quell-Port des empfangenen Datagramms, übereinstimmen. Dabei ist darauf zu achten, daß hier keine Systemports genutzt werden. *xRcvData* bleibt solange gesetzt, wie Eingangsdaten vorhanden sind. *dwRcvPktSrcIP* zeigt den Absender der Daten an und *wRcvLength* zeigt wieviel Daten-Bytes im Puffer, referenziert durch *pbarRcvData*, eingelaufen sind. *wRcvPktSrcPort* sollte normalerweise nicht ausgewertet werden, da dieser Wert, je nach Quell-Port-Bindung, quasi-zufällig sein kann und sich daher von *wRcvPktDstPort* durchaus unterscheiden kann. *wRcvError* signalisiert u.a. Fehler beim Öffnen oder Schließen des Sockels, sowie Datagrammlängenfehler.

Hinweis: Zur Fehleranalyse während der Laufzeit sind die Inhalte der globalen UDP-Variablen in *Global\_UDPcom* evtl. hilfreich.