WAGO I/O SYSTEM 758

Client server applications on the 758-870 using SysLibSocket and WagoLibEthernet_01 library

Application note

A113501, English
Version 1.0.0

WAGO®
INNOVATIVE CONNECTIONS

WAGO Kontakttechnik GmbH
Hansastraße 27
D-32423 Minden

Phone: +49 (0) 571/8 87 – 0
Fax: +49 (0) 571/8 87 – 1 69

E-Mail: info@wago.com

Web: http://www.wago.com

Technical Support
Phone: +49 (0) 571/8 87 – 5 55
Fax: +49 (0) 571/8 87 – 85 55

E-Mail: support@wago.com

Every conceivable measure has been taken to ensure the correctness and completeness of this documentation. However, as errors can never be fully excluded we would appreciate any information or ideas at any time.

We wish to point out that the software and hardware terms as well as the trademarks of companies used and/or mentioned in the present manual are generally trademark or patent protected.

TABLE OF CONTENTS

1 Important comments

To ensure fast installation and start-up of the units described in this manual, we strongly recommend that the following information and explanation is carefully read and adhered to.

1.1 Legal principles

1.1.1 Copyright

This manual is copyrighted, together with all figures and illustrations contained therein. Any use of this manual which infringes the copyright provisions stipulated herein, is not permitted. Reproduction, translation and electronic and photo-technical archiving and amendments require the written consent of WAGO Kontakttechnik GmbH. Non-observance will entail the right of claims for damages.

1.1.2 Personnel qualification

The use of the product detailed in this manual is exclusively geared to specialists having qualifications in PLC programming, electrical specialists or persons instructed by electrical specialists who are also familiar with the valid standards. WAGO Kontakttechnik GmbH declines all liability resulting from improper action and damage to WAGO products and third party products due to non-observance of the information contained in this manual.

1.1.3 Intended use

For each individual application, the components supplied are to work with a dedicated hardware and software configuration. Modifications are only admitted within the framework of the possibilities documented in the manuals. All other changes to the hardware and/or software and the non-conforming use of the components entail the exclusion of liability on part of WAGO Kontakttechnik GmbH.

Please direct any requirements pertaining to a modified and/or new hardware or software configuration directly to WAGO Kontakttechnik GmbH.

1.2 Range of validity

This application note is based on the stated hardware and software of the specific manufacturer as well as the correspondent documentation. This application note is therefore only valid for the described installation.

New hardware and software versions may need to be handled differently. Please note the detailed description in the specific manuals.

1.3 Symbols

Danger
Always observe this information to protect persons from injury.

Warning
Always observe this information to prevent damage to the device.

Attention
Marginal conditions must always be observed to ensure smooth operation.

ESD (Electrostatic Discharge)
Warning of damage to the components by electrostatic discharge. Observe the precautionary measure for handling components at risk.

Note
Routines or advice for efficient use of the device and software optimisation.

More information
References to additional literature, manuals, data sheets and INTERNET pages

2 Description

Use of the library "SysLibSocket" and “WagoLibEthernet_01” for programming client server connections with the Ethernet IPC 758-870.

Four examples will give detailed information on the different steps being necessary to establish a client server connection. The examples will use either TCP or UDP.
A mechanism to avoid blocking mode will be discussed.

3 Reference Material

The connection was tested under the following prerequisites.

Assigned hardware components:	
WAGO IPC	758-870 Release 1
WAGO Switch	758-500

Assigned software components:	
CoDeSys CAA	V2.3.3.6
SysLibSockets.lib	

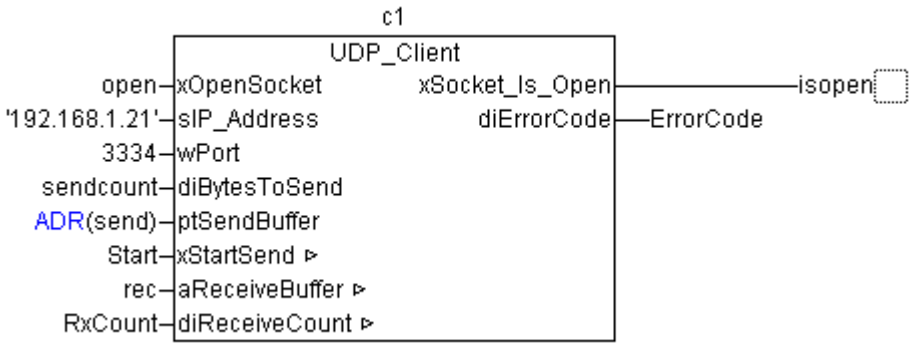
4 Basic function blocks from the library WagoLibEthernet_01

These blocks show the general use of the four function blocks of the WagoLibEthernet_01 library. A detailed functional description can be obtained in the appropriate library description.

4.1 UDP Client

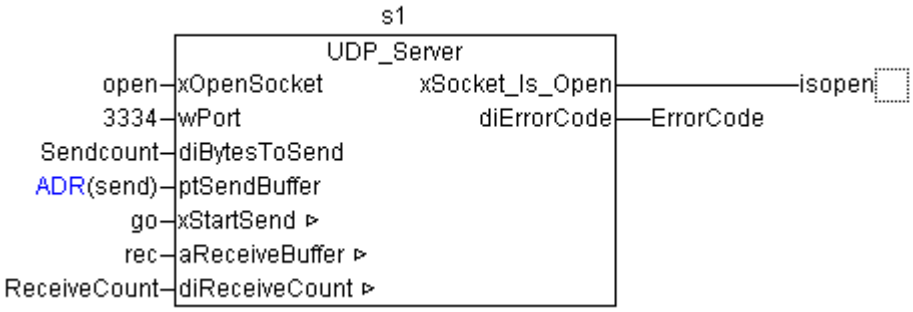
The function block UDP_Client will allow to connect to a UDP server. sIP_Address is the IP address of the server. wPort defines the port for the communication.

By setting the variable xStartSend a message can be transmitted to the server. After execution the function block will reset this variable. Data messages are limited to 1472 Bytes due to the UDP protocol specification.

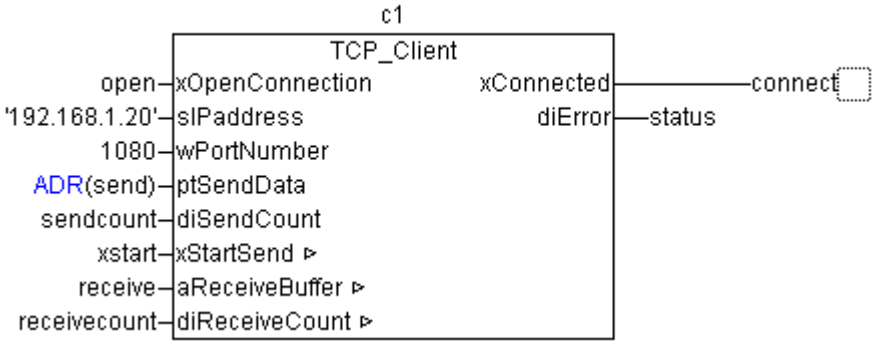


4.2 UDP Server

The function block UDP_Server will allow to act as a UDP server. wPort defines the Port for the communication. After once a message from a client has been received by the server, the server may respond to the client . By setting the variable xStartSend a message can be transmitted to the client. After execution the function block will reset this variable.



4.3 TCP Client



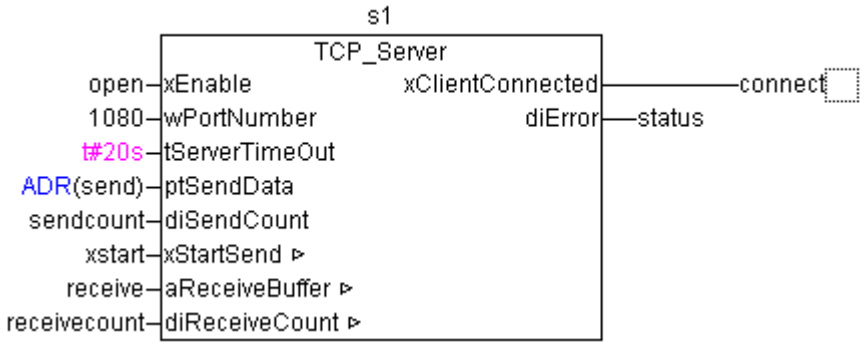
The function block TCP_Client will allow to connect to a TCP server. sIPaddress is the IP address of the server. wPortNumber defines the port for the communication.

First the connection has to be established by the input xOpenConnection. The output xConnected will go high if the connection could be set up. By setting the variable xStartSend a message can be transmitted to the server. After execution the function block will reset this variable.

4.4 TCP Server

The function block TCP_Server will allow to act as a TCP server. wPortNumber defines the Port for the communication.

First the connection has to be established by the client. The output xClientConnected will go high if the server has accepted the connection from a client. By setting the variable xStartSend a message can be transmitted to the client. After execution the function block will reset this variable.

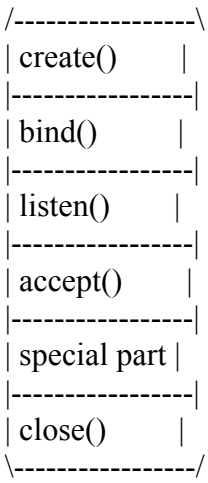

s1
TCP_Server
open
xEnable
1080
wPortNumber
t#20s
tServerTimeOut
ADR(send)
ptSendData
sendcount
diSendCount
xstart
xStartSend
receive
aReceiveBuffer
receivecount
diReceiveCount
xClientConnected
connect
diError
status

5 Theory of client and server programming

There are some general rules for client server connections which are discussed in the following chapters. Since there is a broad range of different client and server applications there is always a special part with each client or server. This document will concentrate on the basic steps.

5.1 How to build a server

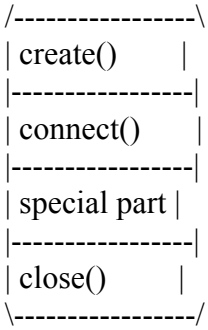
The following drawing shows the basic steps for a server. Each server has at least to create and bind a socket. After that it is necessary to wait for a client to connect to the server. The server meanwhile waits in the accept state for any client.



The server has to take care for closing the connection after the client has closed the appropriate connection.

5.2 How to build a client

The following drawing shows the basic steps for a client. Each client has at least to create a socket and establish the connection to a server by the connect command.



A client should close the connection after the data with the server is exchanged.

5.3 None blocking mode of sockets

A socket is generally created in the blocking mode. This may cause trouble in PLC control since the program will stop and wait. There are different ways to solve this problem.

use one task for socket handling and one task for PLC control

change socket mode to none blocking

Following it will be explained how to change the socket mode. Blocking of sockets will happen with the following functions: SysSockAccept, SysSockConnect, SysSockSend, SysSockSendTo, SysSockRecv, SysSockRecvFrom, SysSockClose.

5.3.1 Using the function SysSockIoCtl

The library SysLibSocket provides the function SysSockIoctl which allows to change the socket mode by the input parameters diCommand and piParameter. In the global variables of the library is a constant(SOCKET_FIONBIO:DINT:=2) defined which is recommended to use in the function call for diCommand. PiParameter is a pointer to a variable which has to be set unequal zero to set the socket to none blocking.


IoCtl does not change the mode for the SysSockConnect. SysSockConnect will always work in blocking mode.

By the following lines a socket can be put to non-blocking mode.

```
m_IoCtlParameter                                    : DINT:= 1;

SysSockIoctl( m_Socket, SOCKET_FIONBIO, ADR(m_IoCtlParameter));
```

5.4 Using the function SysSockRecv

5.4.1 Description

The **SysSockRecv**() function receives a message from a TCP socket. It is normally used with connected sockets because it does not permit the application to retrieve the source address of received data. The function takes the following arguments:

diSocket : Specifies the socket file descriptor.

pbyBuffer : Points to a buffer where the message should be stored.

diBufferSize : Specifies the length in bytes of the buffer

diFlags : Not supported by the implementation

Example:

```
m_BytesReceived := SysSockRecv( diSocket := m_Socket,
                                pbyBuffer := ADR(m_ReceiveBuffer),
                                diBufferSize:= SIZEOF(m_ReceiveBuffer),
                                diFlags:= 0);
```

The **SysSockRecv**() function returns the length of the message written to the buffer pointed to by the pby*Buffer* argument. For stream-based sockets such as **SOCK_STREAM**, message boundaries are ignored. In this case, data is returned to the user as soon as it becomes available, and no data is discarded.

5.4.2 Return value

Upon successful completion, **SockRecv** () returns the length of the message in bytes

If no messages are available at the socket and the socket is not set to blocking **Sys-SockRecv** () blocks until a message arrives. If no messages are available at the socket and the socket is set to non-blocking –1 is returned.

If no messages are available to be received and the peer has performed an orderly shut-down, **SockRecv** () returns 0(Gracefully closed).

5.5 Using the function SysSockSend

5.5.1 Description

The **SysSockSend**() function transmits a message through a TCP socket. The function takes the following arguments:

diSocket : Specifies the socket file descriptor.

pbyBuffer : Points to a buffer where the message should be stored.

diBufferSize : Specifies the length in bytes of the buffer

diFlags : Not supported by the implementation

Example:

```
m_diReturn := SysSockSend(    diSocket := m_Socket,
                              pbyBuffer := ADR(m_TxBuffer),
                              diBufferSize:= SIZEOF(m_TxBuffer),
```

```
diFlags:= 0);
```

5.5.2 Return value

Upon successful completion, **SysSockSend** () returns the length of the message in bytes.

If an error occurred –1 will be returned.

If no messages have to be send and the peer has performed an orderly shutdown, **SysSockSend** () returns 0.

5.6 Using the function SysSockRecvFrom

The **SysSockRecvFrom**() function receives a message from a UDP socket. This function allows the application to retrieve the source address of the transmitter. The address is copied to the variable pointed by pSockAddr. If no data are received this area will be zero. The function takes the following arguments:

diSocket :	Specifies the socket file descriptor.
pbyBuffer :	Points to a buffer where the message should be stored.
diBufferSize :	Specifies the length in bytes of the buffer
diFlags :	Not supported by the implementation
pSockAddr:	Points to a buffer where the address of the transmitter is stored
diSockAddrSize	Specifies the length of the address buffer

Example:

```
SysSockRecvFrom(diSocket:= m_Socket,
                pbyBuffer := ADR(m_ReceiveBuffer),(* Address of receive buffer *)
                diBufferSize      := SIZEOF(m_ReceiveBuffer),
                diFlags           := 0,
                pSockAddr         := ADR( m_AddressInfo),
```

```
                diSockAddrSize := SIZEOF(m_AddressInfo));
```

5.6.1 Return value

Upon successful completion, **SysSockRecvFrom** () returns the length of the message in bytes.

If no messages are available at the socket and the socket is set to non-blocking -1 is returned.

5.7 Using the function SysSockSendTo

The **SysSockSendTo**() function transmits a message through a UDP socket. The address of the destination has to be placed in the area pointed by pSockAddr. If no data are received this area will be zero. The function takes the following arguments:

diSocket : Specifies the socket file descriptor.

pbyBuffer : Points to a buffer where the message should be stored.

diBufferSize : Specifies the length in bytes of the buffer

diFlags : Not supported by the implementation

pSockAddr: Points to a buffer where the address of the transmitter is stored

diSockAddrSize Specifies the length of the address buffer

Example:

```
SysSockSendTo(diSocket:= m_Socket,
                pbyBuffer := ADR(m_SendBuffer),(* Address of transmit buffer *)
                diBufferSize      := SIZEOF(m_SendBuffer),
                diFlags           := 0,
                pSockAddr         := ADR( m_AddressInfo),
                diSockAddrSize := SIZEOF(m_AddressInfo));
```

5.7.1 Return value

Upon successful completion, **SysSockSendTo** () returns the length of the message in bytes.

If no data could be transmitted –1 will be returned.

WAGO Kontakttechnik GmbH
Postfach 2880 • D-32385 Minden
Hansastraße 27 • D-32423 Minden
Phone: 05 71/8 87 – 0
Telefax: 05 71/8 87 – 1 69
E-Mail: info@wago.com

Internet: http://www.wago.com