SoMachine
TcpUdpCommunication Library Guide

06/2017

EIO0000002204.02

www.schneider-electric.com

Schneider Electric

The information provided in this documentation contains general descriptions and/or technical characteristics of the performance of the products contained herein. This documentation is not intended as a substitute for and is not to be used for determining suitability or reliability of these products for specific user applications. It is the duty of any such user or integrator to perform the appropriate and complete risk analysis, evaluation and testing of the products with respect to the relevant specific application or use thereof. Neither Schneider Electric nor any of its affiliates or subsidiaries shall be responsible or liable for misuse of the information contained herein. If you have any suggestions for improvements or amendments or have found errors in this publication, please notify us.

All pertinent state, regional, and local safety regulations must be observed when installing and using this product. For reasons of safety and to help ensure compliance with documented system data, only the manufacturer should perform repairs to components.

When devices are used for applications with technical safety requirements, the relevant instructions must be followed.

Failure to use Schneider Electric software or approved software with our hardware products may result in injury, harm, or improper operating results.

Failure to observe this information can result in injury or equipment damage.

Table of Contents

Safety Information

Important Information

NOTICE

Read these instructions carefully, and look at the equipment to become familiar with the device before trying to install, operate, service, or maintain it. The following special messages may appear throughout this documentation or on the equipment to warn of potential hazards or to call attention to information that clarifies or simplifies a procedure.

The addition of this symbol to a “Danger” or “Warning” safety label indicates that an electrical hazard exists which will result in personal injury if the instructions are not followed.

This is the safety alert symbol. It is used to alert you to potential personal injury hazards. Obey all safety messages that follow this symbol to avoid possible injury or death.

⚠ DANGER

DANGER indicates a hazardous situation which, if not avoided, **will result in** death or serious injury.

⚠ WARNING

WARNING indicates a hazardous situation which, if not avoided, **could result in** death or serious injury.

⚠ CAUTION

CAUTION indicates a hazardous situation which, if not avoided, **could result** in minor or moderate injury.

NOTICE

NOTICE is used to address practices not related to physical injury.

PLEASE NOTE

Electrical equipment should be installed, operated, serviced, and maintained only by qualified personnel. No responsibility is assumed by Schneider Electric for any consequences arising out of the use of this material.

A qualified person is one who has skills and knowledge related to the construction and operation of electrical equipment and its installation, and has received safety training to recognize and avoid the hazards involved.

BEFORE YOU BEGIN

Do not use this product on machinery lacking effective point-of-operation guarding. Lack of effective point-of-operation guarding on a machine can result in serious injury to the operator of that machine.

⚠ WARNING

UNGUARDED EQUIPMENT

- Do not use this software and related automation equipment on equipment which does not have point-of-operation protection.
- Do not reach into machinery during operation.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

This automation equipment and related software is used to control a variety of industrial processes. The type or model of automation equipment suitable for each application will vary depending on factors such as the control function required, degree of protection required, production methods, unusual conditions, government regulations, etc. In some applications, more than one processor may be required, as when backup redundancy is needed.

Only you, the user, machine builder or system integrator can be aware of all the conditions and factors present during setup, operation, and maintenance of the machine and, therefore, can determine the automation equipment and the related safeties and interlocks which can be properly used. When selecting automation and control equipment and related software for a particular application, you should refer to the applicable local and national standards and regulations. The National Safety Council's Accident Prevention Manual (nationally recognized in the United States of America) also provides much useful information.

In some applications, such as packaging machinery, additional operator protection such as point-of-operation guarding must be provided. This is necessary if the operator's hands and other parts of the body are free to enter the pinch points or other hazardous areas and serious injury can occur. Software products alone cannot protect an operator from injury. For this reason the software cannot be substituted for or take the place of point-of-operation protection.

Ensure that appropriate safeties and mechanical/electrical interlocks related to point-of-operation protection have been installed and are operational before placing the equipment into service. All interlocks and safeties related to point-of-operation protection must be coordinated with the related automation equipment and software programming.

NOTE: Coordination of safeties and mechanical/electrical interlocks for point-of-operation protection is outside the scope of the Function Block Library, System User Guide, or other implementation referenced in this documentation.

START-UP AND TEST

Before using electrical control and automation equipment for regular operation after installation, the system should be given a start-up test by qualified personnel to verify correct operation of the equipment. It is important that arrangements for such a check be made and that enough time is allowed to perform complete and satisfactory testing.

⚠ WARNING

EQUIPMENT OPERATION HAZARD

- Verify that all installation and set up procedures have been completed.
- Before operational tests are performed, remove all blocks or other temporary holding means used for shipment from all component devices.
- Remove tools, meters, and debris from equipment.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Follow all start-up tests recommended in the equipment documentation. Store all equipment documentation for future references.

Software testing must be done in both simulated and real environments.

Verify that the completed system is free from all short circuits and temporary grounds that are not installed according to local regulations (according to the National Electrical Code in the U.S.A, for instance). If high-potential voltage testing is necessary, follow recommendations in equipment documentation to prevent accidental equipment damage.

Before energizing equipment:

- Remove tools, meters, and debris from equipment.
- Close the equipment enclosure door.
- Remove all temporary grounds from incoming power lines.
- Perform all start-up tests recommended by the manufacturer.

OPERATION AND ADJUSTMENTS

The following precautions are from the NEMA Standards Publication ICS 7.1-1995 (English version prevails):

- Regardless of the care exercised in the design and manufacture of equipment or in the selection and ratings of components, there are hazards that can be encountered if such equipment is improperly operated.
- It is sometimes possible to misadjust the equipment and thus produce unsatisfactory or unsafe operation. Always use the manufacturer's instructions as a guide for functional adjustments. Personnel who have access to these adjustments should be familiar with the equipment manufacturer's instructions and the machinery used with the electrical equipment.
- Only those operational adjustments actually required by the operator should be accessible to the operator. Access to other controls should be restricted to prevent unauthorized changes in operating characteristics.

About the Book

At a Glance

Document Scope

This document describes the library TcpUdpCommunication.

The library provides the core functionality for implementing socket-based network communication protocols using TCP (Transmission Control Protocol) client and server, or UDP (User Datagram Protocol) including broadcast and multicast if supported by the platform. Only communication based on IPv4 via the Ethernet-ports of the controller is supported.

The TcpUdpCommunication library uses system functions and resources which are supported on specific controller platforms available in SoMachine.

The following controllers are supported:
- Modicon M251 Logic Controller
- Modicon M241 Logic Controller (must be equipped with an Ethernet interface, can be either embedded Ethernet or with the TM4ES4 expansion module)
- Modicon M258 Logic Controller
- Modicon LMC058 Motion Controller
- Modicon LMC078 Motion Controller

Validity Note

This document has been updated for the release of SoMachine V4.3.

The technical characteristics of the devices described in this document also appear online. To access this information online:

Step	Action
1	Go to the Schneider Electric home page *www.schneider-electric.com*.
2	In the **Search** box type the reference of a product or the name of a product range. ● Do not include blank spaces in the reference or product range. ● To get information on grouping similar modules, use asterisks (*).
3	If you entered a reference, go to the **Product Datasheets** search results and click on the reference that interests you. If you entered the name of a product range, go to the **Product Ranges** search results and click on the product range that interests you.
4	If more than one reference appears in the **Products** search results, click on the reference that interests you.
5	Depending on the size of your screen, you may need to scroll down to see the data sheet.
6	To save or print a data sheet as a .pdf file, click **Download XXX product datasheet**.

The characteristics that are presented in this manual should be the same as those characteristics that appear online. In line with our policy of constant improvement, we may revise content over time to improve clarity and accuracy. If you see a difference between the manual and online information, use the online information as your reference.

Terminology Derived from Standards

The technical terms, terminology, symbols and the corresponding descriptions in this manual, or that appear in or on the products themselves, are generally derived from the terms or definitions of international standards.

In the area of functional safety systems, drives and general automation, this may include, but is not limited to, terms such as *safety*, *safety function*, *safe state*, *fault*, *fault reset*, *malfunction*, *failure*, *error*, *error message*, *dangerous*, etc.

Among others, these standards include:

Standard	Description
EN 61131-2:2007	Programmable controllers, part 2: Equipment requirements and tests.
ISO 13849-1:2008	Safety of machinery: Safety related parts of control systems. General principles for design.
EN 61496-1:2013	Safety of machinery: Electro-sensitive protective equipment. Part 1: General requirements and tests.
ISO 12100:2010	Safety of machinery - General principles for design - Risk assessment and risk reduction
EN 60204-1:2006	Safety of machinery - Electrical equipment of machines - Part 1: General requirements
EN 1088:2008 ISO 14119:2013	Safety of machinery - Interlocking devices associated with guards - Principles for design and selection
ISO 13850:2006	Safety of machinery - Emergency stop - Principles for design
EN/IEC 62061:2005	Safety of machinery - Functional safety of safety-related electrical, electronic, and electronic programmable control systems
IEC 61508-1:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: General requirements.
IEC 61508-2:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: Requirements for electrical/electronic/programmable electronic safety-related systems.
IEC 61508-3:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: Software requirements.
IEC 61784-3:2008	Digital data communication for measurement and control: Functional safety field buses.
2006/42/EC	Machinery Directive
2014/30/EU	Electromagnetic Compatibility Directive
2014/35/EU	Low Voltage Directive

In addition, terms used in the present document may tangentially be used as they are derived from other standards such as:

Standard	Description
IEC 60034 series	Rotating electrical machines
IEC 61800 series	Adjustable speed electrical power drive systems
IEC 61158 series	Digital data communications for measurement and control – Fieldbus for use in industrial control systems

Finally, the term *zone of operation* may be used in conjunction with the description of specific hazards, and is defined as it is for a *hazard zone* or *danger zone* in the *Machinery Directive* (*2006/42/EC*) and *ISO 12100:2010*.

NOTE: The aforementioned standards may or may not apply to the specific products cited in the present documentation. For more information concerning the individual standards applicable to the products described herein, see the characteristics tables for those product references.

Part I
General Information

What Is in This Part?

This part contains the following chapters:

Chapter 1
Specific Safety Information

Overview

This section contains information regarding working with the TcpUdpCommunication library. Qualified personnel working with the TcpUdpCommunication library must read and observe this information.

What Is in This Chapter?

This chapter contains the following topics:

Qualification of Personnel

Overview

Electrical equipment should be installed, operated, serviced, and maintained only by qualified personnel.

No responsibility is assumed by Schneider Electric for any consequences arising out of the use of this material.

A qualified person is one who has skills and knowledge related to the construction and operation of electrical equipment and the installation, and has received safety-related training to recognize and avoid the hazards involved.

The qualified person must be able to detect possible hazards that may arise from parameterization, modifying parameter values and generally from mechanical, electrical, or electronic equipment. The qualified person must be familiar with the standards, provisions, and regulations for the prevention of industrial accidents, which they must observe when designing and implementing the system.

Proper Use

Overview

This product is a library to be used together with the control systems and servo amplifiers intended solely for the purposes as described in the present documentation as applied in the industrial sector.

Always observe the applicable safety-related instructions, the specified conditions, and the technical data.

Perform a risk evaluation concerning the specific use before using the product. Take protective measures according to the result.

Since the product is used as a part of an overall system, you must ensure the safety of the personnel by means of the concept of this overall system (for example, machine concept).

Any other use is not intended and may be hazardous. Electrical devices and equipment must only be installed, operated, maintained, and repaired by qualified personnel.

Product Related Information

Product Related Information

⚠ WARNING

LOSS OF CONTROL

- The designer of any control scheme must consider the potential failure modes of control paths and, for certain critical control functions, provide a means to achieve a safe state during and after a path failure. Examples of critical control functions are emergency stop and overtravel stop, power outage and restart.
- Separate or redundant control paths must be provided for critical control functions.
- System control paths may include communication links. Consideration must be given to the implications of unanticipated transmission delays or failures of the link.
- Observe all accident prevention regulations and local safety guidelines.[1]
- Each implementation of this equipment must be individually and thoroughly tested for proper operation before being placed into service.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

[1] For additional information, refer to NEMA ICS 1.1 (latest edition), "Safety Guidelines for the Application, Installation, and Maintenance of Solid State Control" and to NEMA ICS 7.1 (latest edition), "Safety Standards for Construction and Guide for Selection, Installation and Operation of Adjustable-Speed Drive Systems" or their equivalent governing your particular location.

Before you attempt to provide a solution (machine or process) for a specific application using the POUs found in the library, you must consider, conduct and complete best practices. These practices include, but are not limited to, risk analysis, functional safety, component compatibility, testing and system validation as they relate to this library.

⚠ WARNING

IMPROPER USE OF POUS

- Perform a safety-related analysis for the application and the devices installed.
- Ensure that the POUs are compatible with the devices in the system and have no unintended effects on the proper functioning of the system.
- Use appropriate parameters, especially limit values, and observe machine wear and stop behavior.
- Verify that the sensors and actuators are compatible with the selected POUs.
- Thoroughly test all functions during verification and commissioning in all operation modes.
- Provide independent methods for critical control functions (emergency stop, conditions for limit values being exceeded, etc.) according to a safety-related analysis, respective rules, and regulations.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Incomplete file transfers, such as data files, application files and/or firmware files, may have serious consequences for your machine or controller. If you remove power, or if there is a power outage or communication interruption during a file transfer, your machine may become inoperative, or your application may attempt to operate on a corrupted data file. If a an interruption occurs, reattempt the transfer. Be sure to include in your risk analysis the impact of corrupted data files.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION, DATA LOSS, OR FILE CORRUPTION

- Do not interrupt an ongoing data transfer.
- If the transfer is interrupted for any reason, re-initiate the transfer.
- Do not place your machine into service until the file transfer has completed successfully, unless you have accounted for corrupted files in your risk analysis and have taken appropriate steps to prevent any potentially serious consequences due to unsuccessful file transfers.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Chapter 2
Presentation of the Library

General Information

Library Overview

The library provides the core functionality for implementing socket-based network communication protocols using TCP (Transmission Control Protocol) client and server, or UDP (User Datagram Protocol) including broadcast and multicast if supported by the platform. Only communication based on IPv4 via the Ethernet-ports of the controller is supported.

You have to implement the application protocol used on the remote site (which can be hardware such as barcode scanners, vision cameras, industrial robots, or computer systems running software like database servers).

The user interface of the library is derived from BSD-style socket implementations comparable to the ones used when programming communication applications under UNIX or Microsoft Windows based systems. It uses these main functions:

- `Open` / `Bind` / `Connect`
  to initialize the communication
- `Accept`
  to accept incoming TCP-connections when running a TCP server
- `Close` / `Disconnect` / `Shutdown`
  to end the communication
- `Send` / `SendTo`
  to transmit data to remote systems
- `Receive` / `ReceiveFrom`
  to process data sent by remote systems
- `SocketOptions`
  to tune the low-level behavior of the communication

In this library, an object-oriented approach has been chosen to allow a clean design of the application program. Thus the functions presented above are available as methods and properties of the respective function blocks explained in detail in chapter Program Organization Units (POU) *(see page 47)*.

Characteristic	Value
Library title	TcpUdpCommunication
Company	Schneider Electric
Category	**Communication**
Component	**Core Repository**
Default namespace	**TCPUDP**
Language model attribute	qualified-access-only *(see SoMachine, Functions and Libraries User Guide)*
Forward compatible library	Yes (FCL *(see SoMachine, Functions and Libraries User Guide)*)

NOTE: For this library, qualified-access-only is set. This means, that the POUs, data structures, enumerations, and constants have to be accessed using the namespace of the library. The default namespace of the library is **TCPUDP**.

Example Project

In conjunction with the library the example project TcpUdpCommunicationExample.project is provided. The example project shows how to implement a data exchange between two controllers over the Ethernet network with using the TcpUdpCommunication library.

The example project is installed on your PC along with the SoMachine software.

To open the project example, proceed as follows:

Step	Action
1	Launch SoMachine Central on your PC by double-clicking the respective icon on your desktop.
2	In the SoMachine Central **Get Started** screen, click the **Help Center** button at the top right.
3	Click the **Examples** button.
4	Double-click the **Communication** folder icon.
5	Double-click the **TcpUdpCommunication** folder icon.
6	Double-click the **TcpUdpCommunicationExample.project** project icon. **Result**: A copy of the project example is open in a new instance of SoMachine.

General Considerations

Only IPv4 IP addresses are supported for the communication functions provided with this library.

The TcpUdpCommunication (Schneider Electric) and the CAA Net Base Services library (CAA Technical Workgroup) use the same system resources on the controller. The simultaneous use of both libraries in the same application may lead to disturbances during the operation of the controller.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

Do not use the library TcpUdpCommunication (Schneider Electric) together with the library CAA Net Base Services (CAA Technical Workgroup) simultaneously in the same application.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Consideration Concerning Cyber Security

The functions provided with this library do not support secure connections such as TLS (Transport Layer Security) or SSL (Secure Socket Layer). Since the emails are not encrypted, a specific email server is required for this communication. Communication must only be performed inside your industrial network, isolated from other networks inside your company, and protected from the Internet.

NOTE: Schneider Electric adheres to industry best practices in the development and implementation of control systems. This includes a "Defense-in-Depth" approach to secure an Industrial Control System. This approach places the controllers behind one or more firewalls to restrict access to authorized personnel and protocols only.

⚠ WARNING

UNAUTHENTICATED ACCESS AND SUBSEQUENT UNAUTHORIZED MACHINE OPERATION

- Evaluate whether your environment or your machines are connected to your critical infrastructure and, if so, take appropriate steps in terms of prevention, based on Defense-in-Depth, before connecting the automation system to any network.
- Limit the number of devices connected to a network to the minimum necessary.
- Isolate your industrial network from other networks inside your company.
- Protect any network against unintended access by using firewalls, VPN, or other, proven security measures.
- Monitor activities within your systems.
- Prevent subject devices from direct access or direct link by unauthorized parties or unauthenticated actions.
- Prepare a recovery plan including backup of your system and process information.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Overview of the POUs

Function block	Use
FB_TCPClient *(see page 53)*	Implements a TCP client, which connects to a TCP server to allow data exchange.
FB_TCPServer *(see page 76)*	Implements a TCP server, which listens for and processes incoming client-connections on a specified port.
FB_UDPPeer *(see page 110)*	Implements a UDP peer, which represents an endpoint for sending and receiving messages using the message-based UDP protocol.

Overview of the Structures in the Module-Specific Interface

Structure	Use
ST_ClientConnection *(see page 34)*	Contains information about the clients connected to the server.
ST_DefaultSocketOptionsTCPClient *(see page 35)*	Contains information about which socket options should be automatically set or modified when a new connection is established.
ST_DefaultSocketOptionsTCPServer *(see page 36)*	Contains information about which socket options should be automatically set or modified when a new server instance is opened.
ST_DefaultSocketOptionsUDPPeerST_DefaultSocketOptionsUDPPeer *(see page 37)*	Contains information about which socket options should be automatically set or modified when a new peer instance is opened.

Overview of the Enumerations

Enumeration	Use
ET_Result *(see page 28)*	Contains the possible values that indicate the result of operations executed by the function block.
ET_State *(see page 31)*	Contains the possible values that indicate the state of the function blocks.

Part II
Enumerations and Structures

What Is in This Part?

This part contains the following chapters:

Chapter 3
Enumerations

What Is in This Chapter?

This chapter contains the following topics:

ET_Result

Overview

Type:	Enumeration
Available as of:	V1.0.9.0

Description

The enumeration `ET_Result` contains the possible values that indicate the result of operations executed by the function block.

Enumeration Elements

The values 0 to 99 are used to indicate the result of the operations executed by the function blocks `FB_TcpClient`, `FB_TcpServer`, and `FB_UdpPeer`.

Name	Value	Description
`Ok`	0	The operation completed successfully.
`Failed`	1	The operation was not completed successfully.
`NotReady`	10	The requested operation cannot be executed in the present state or the property `Result` was different than `Ok` prior to the function call.
`NotSupported`	11	The requested operation is not supported by this controller.
`AddressInUse`	12	The requested port address is already in use.
`AddressNotAvailable`	13	The requested IP address is not available since it is not configured on any Ethernet interface.
`InputOutOfRange`	20	The value is outside of the valid range.
`ReceiveBufferSizeOutOfRange`	21	The value of the input specifying the receive buffer size is outside of the valid range.
`NumBytesToSendOutOfRange`	22	The value of the input specifying the number of bytes to send is outside of the valid range.
`FillLevelOutOfRange`	23	The value of the input specifying the fill level of the buffer is outside of the valid range.
`TooMuchOOBData`	24	Only 1 byte of `OutOfBand`-data is allowed but a larger size was requested.
`InvalidBufferAddress`	25	The address of the buffer is invalid.
`InvalidIP`	26	The given IP address is not valid.
`InvalidMulticastIP`	27	The given IP address is not valid or not in the range of multicast addresses.

Name	Value	Description
NoSuchClient	28	The client specified could not be found since it is not connected to the TCP server.
ClosedByPeer	40	The requested operation cannot be executed since the remote site has already closed the connection.
ConnectionTimedOut	41	The attempt to establish a connection has timed out because the remote system is not available or is not answering the request.
ConnectionRefused	42	The attempt to establish a connection is unsuccessful because the remote system denied it.
NotEnoughResources	50	The requested operation cannot be executed because there are not enough configurable internal resources available. Try using a smaller amount of data for 1 call.
SocketManagementListTooSmall	51	The socket cannot be opened since the size of the internal socket management list is insufficient. Close an already open socket or increase the value of Gc_uiSocketManagementListSize in the global parameter list (GPL).
ClientListTooSmall	52	The TCP server cannot have any more clients. Disconnect an existing client or increase the value of Gc_uiTCPServerMaxConnections in the global parameter list (GPL).
SendToAllSizeTooSmall	53	The requested number of bytes to send exceeds the limit. Reduce the amount of data to be sent or increase the value of Gc_udiTCPServerMaxSendToAllSize in the global parameter list (GPL).
BufferFull	54	The send buffer of the TCP stack is full, no more data can be copied. Increase send buffer size using the respective socket option or try to send again later.

The values 1001 to 1100 are used to indicate the status of the function block FB_DnsClient. If q_xError of the function block is FALSE, one of the following status messages is provided through the q_etResult.

Name	Value	Description
Disabled	1001	The function block is disabled.
Initializing	1002	The function block is initializing.
Disabling	1003	The function block is disabling.
Ready	1004	The function block is ready.
SendDnsQuery	1010	The function block is sending query to DNS server.
WaitForDnsAnswer	1012	The function block is waiting for answer from DNS server.
AnalyzeDnsAnswer	1014	The function block is analyzing the answer.

The values 1500 to 1600 are used to indicate the operations executed by the function block FB_DnsClient. If q_xError of the function block is TRUE, one of the following error messages is provided through the q_etResult.

Name	Value	Description
InvalidDnsServerIP	1500	The specified IP address of the DNS server is invalid. **NOTE:** i_sDnsServerIP must be something other than null and of the correct format.
InvalidDomainName	1502	No or invalid domain name specified.
OpenSocketFailed	1506	An error has been detected while opening an UDP socket.
SendDnsQueryFailed	1508	An error was detected while sending the query to the DNS server.
BufferSizeTooSmall	1510	Buffer size defined by GPL.Gc_udiDnsBufferSize is insufficient.
ReceiveDnsAnswerFailed	1512	An error was detected while receiving the data from the DNS server.
InvalidDnsAnswer	1514	DNS answer from server is invalid.
InvalidNumberOfIPs	1516	The value for the parameter GPL.Gc_udiDnsNumberOfIPs cannot be 0.
DnsResolutionFailed	1518	The DNS server could not resolve the requested domain name.
InternalError	1525	Internal error.
DnsServerError	1526	The DNS server response contained an error code. Refer to the output q_sResultMsg of the FB_DnsClient.
InvalidDnsTimeOut	1527	The value for the parameter GPL.Gc_timDnsTimeOut cannot be 0.

Used By

- FB_Dns_Client
- FB_TCPClient
- FB_TCPServer
- FB_UDPPeer
- FC_EtResultToString

ET_State

Overview

Type:	Enumeration
Available as of:	V1.0.4.0

Description

The enumeration `ET_State` contains the possible values that indicate the state of the function blocks.

Enumeration Elements

Name	Value	Description
`Idle`	0	Default state, ready for `Connect` (`FB_TCPClient`) or `Open` (`FB_TCPServer` and `FB_UDPPeer`).
`Connecting`	10	The attempt to establish a connection has been started but no result is available yet.
`Connected`	11	The connection has been established successfully and data transfer is possible.
`Shutdown`	12	The connection has been closed by the remote site and no more data can be sent.
`Listening`	20	The server has been started successfully and is waiting for incoming connections.
`Opened`	30	The UDP socket has been opened but is not yet bound to a local port. Sending of messages or binding it to a port is possible.
`Bound`	31	The UDP socket has been bound to a local port. Messages can be sent and received.

Used By

- `FB_TCPClient`
- `FB_TCPServer`
- `FB_UDPPeer`
- `FC_EtStateToString`

Chapter 4
Structures

What Is in This Chapter?

This chapter contains the following topics:

ST_ClientConnection

Overview

Type:	Structure
Available as of:	V1.0.4.0
Inherits from:	-

Description

The structure ST_ClientConnection contains information about the clients connected to the server.

Structure Elements

Name	Data type	Description
sClientIP	STRING(15)	The IP address of the connected client
uiClientPort	UINT	The port number of the connected client
udiBytesAvailableToRead	UDINT	Number of bytes that are available to be read from that client.
uliTotalBytesReceived	ULINT	Total number of bytes received from this client.
uliTotalBytesSent	ULINT	Total number of bytes sent to this client.

Used By

- FB_TCPServer

ST_DefaultSocketOptionsTCPClient

Overview

Type:	Structure
Available as of:	V1.0.4.0
Inherits from:	-

Description

The structure ST_DefaultSocketOptionsTCPClient contains information about which socket options should be automatically set or modified when a new connection is established. For a description of the specific socket options, see the description of the property.

Structure Elements

Name	Data type	Default value	Description
xModifyKeepAlive	BOOL	TRUE	If TRUE, sets the socket option KeepAlive to the value of xKeepAliveValue.
xKeepAliveValue	BOOL	TRUE	Value to set socket option KeepAlive to.
xModifyNoDelay	BOOL	FALSE	If TRUE, sets the socket option NoDelay to the value of xNoDelayValue.
xNoDelayValue	BOOL	TRUE	Value to set socket option NoDelay to.
xModifySendBufferSize	BOOL	FALSE	If TRUE, sets the socket option SendBufferSize to the value of udiSendBufferSizeValue.
udiSendBufferSizeValue	UDINT	10000	Value to set socket option SendBufferSize to.
xModifyReceiveBufferSize	BOOL	FALSE	If TRUE, sets the socket option ReceiveBufferSize to the value of udiReceiveBufferSizeValue.
udiReceiveBufferSizeValue	UDINT	10000	Value to set socket option ReceiveBufferSize to.
xModifyOutOfBandInline	BOOL	FALSE	If TRUE, sets the socket option OutOfBandInline to the value of xOutOfBandInlineValue.
xOutOfBandInlineValue	BOOL	TRUE	Value to set socket option OutOfBandInline to.

Used By

- FB_TCPClient

ST_DefaultSocketOptionsTCPServer

Overview

Type:	Structure
Available as of:	V1.0.8.0
Inherits from:	-

Description

The structure ST_DefaultSocketOptionsTCPServer contains information about which socket options should be automatically set or modified when a new server instance is opened.

Structure Elements

Name	Data type	Default value	Description
xModifyKeepAlive	BOOL	TRUE	If TRUE, sets the socket option KeepAlive to the value of xKeepAliveValue.
xKeepAliveValue	BOOL	TRUE	Value to set socket option KeepAlive to.
xModifyReuseAddress	BOOL	TRUE	If TRUE, sets the socket option ReuseAddress to the value of xReuseAddressValue.
xReuseAddressValue	BOOL	TRUE	Value to set socket option ReuseAddress to.
xModifySendBufferSize	BOOL	FALSE	If TRUE, sets the socket option SendBufferSize to the value of udiSendBufferSizeValue.
udiSendBufferSizeValue	UDINT	10000	Value to set socket option SendBufferSize to.
xModifyReceiveBufferSize	BOOL	FALSE	If TRUE, sets the socket option ReceiveBufferSize to the value of udiReceiveBufferSizeValue.
udiReceiveBufferSizeValue	UDINT	10000	Value to set socket option ReceiveBufferSize to.

Used By

- FB_TCPServer

ST_DefaultSocketOptionsUDPPeer

Overview

Type:	Structure
Available as of:	V1.0.4.0
Inherits from:	-

Description

The structure ST_DefaultSocketOptionsUDPPeer contains information about which socket options should be automatically set or modified when a new peer instance is opened.

Structure Elements

Name	Data type	Default value	Description
xModifySendBufferSize	BOOL	FALSE	If TRUE, sets the socket option SendBufferSize to the value of udiSendBufferSizeValue.
udiSendBufferSizeValue	UDINT	10000	Value to set socket option SendBufferSize to.
xModifyReceiveBufferSize	BOOL	FALSE	If TRUE, sets the socket option ReceiveBufferSize to the value of udiReceiveBufferSizeValue.
udiReceiveBufferSizeValue	UDINT	10000	Value to set socket option ReceiveBufferSize to.
xModifyBroadcast	BOOL	TRUE	If TRUE, sets the socket option Broadcast to the value of xBroadcastValue.
xBroadcastValue	BOOL	TRUE	Value to set socket option Broadcast to.
xModifyMulticastDefaultInterface	BOOL	FALSE	If TRUE, sets the socket option MulticastDefaultInterface to the value of sMulticastDefaultInterfaceValue.
sMulticastDefaultInterfaceValue	STRING (15)	'' (empty string)	Value to set socket option MulticastDefaultInterface to.
xModifyMulticastDefaultTimeToLive	BOOL	FALSE	If TRUE, sets the socket option MulticastDefaultTimeToLive to the value of siMulticastDefaultTimeToLiveValue.
siMulticastDefaultTimeToLiveValue	SINT	16	Value to set socket option MulticastDefaultTimeToLive to.

Name	Data type	Default value	Description
xModifyMulticastLoopback	BOOL	FALSE	If TRUE, sets the socket option MulticastLoopback to the value of xMulticastLoopbackValue.
xMulticastLoopbackValue	BOOL	FALSE	Value to set socket option MulticastLoopback to.

Used By

- FB_UDPPeer

ST_DnsAdressInfo

Overview

Type:	Structure
Available as of:	V1.0.4.0
Inherits from:	-

Description

The structure `ST_DnsAddressInfo` contains information about the resolved domain name received from the DNS server.

Structure Elements

Name	Data type	Description
`sIpAddress`	STRING(15)	The IP address of the resolved domain name.
`dwTTL`	DWORD	The time to live (TTL) of the IP address. (Time in seconds indicating how long the IP address can be cached.)

Part III
Global Variables

What Is in This Part?

This part contains the following chapters:

Chapter 5
Global Parameter List

GPL

Overview

Type:	Global parameters
Available as of:	V1.0.4.0

Description

The global parameter list (GPL) contains the global parameters of the TcpUdpCommunication library. The parameters can be edited in the library info window of the **Library Manager**.

Global Parameters

Variable	Data type	Value	Description
`Gc_uiTCPServerMaxConnections`	UINT	30	Maximum of total connections one instance of `FB_TCPServer` handles.
`Gc_uiTCPServerMaxBacklog`	UINT	5	Maximum number of incoming (but not yet accepted) connections one instance of `FB_TCPServer` handles.
`Gc_udiTCPServerMaxSendToAllSize`	UDINT	2000	Maximum number of bytes to be sent using the `SendToAll`-method.
`Gc_uiSocketManagementListSize`	UINT	200	Total maximum of sockets supported by the library (sum of the UDP, TCP client, and TCP server sockets).
`Gc_uiDnsBufferSize`	UINT	512	Size (in bytes) of the send and receive buffer used by the `FB_DnsClient`. **NOTE:** Decreasing the default value may result in error message `ET_Result.BufferSizeTooSmall`.
`Gc_timDnsTimeOut`	TIME	T#10s	Timeout for waiting for a response from the DNS server
`Gc_uiDnsNumberOfIPs`	UINT	1	Number of IP addresses to be returned from DNS server.

Chapter 6
Global Variable List

GVL

Overview

Type:	Global variables
Available as of:	V1.0.4.0

Description

The global variable list contains the global variables of the TcpUdpCommunication library.

The global variables are automatically used by the respective function blocks.

Global Variables

Variable	Data type	Description
`G_stDefaultSocketOptionsTCPClient`	ST_DefaultSocketOptionsTCPClient	Default value for socket options that are set when a TCP client connection is initialized.
`G_stDefaultSocketOptionsTCPServer`	ST_DefaultSocketOptionsTCPServer	Default value for socket options that are set when a TCP server socket is initialized.
`G_stDefaultSocketOptionsUDPPeer`	ST_DefaultSocketOptionsUDPPeer	Default value for socket options that are set when a UDP socket is initialized.

Code Example

The code example shows how to change the default socket option for the TCP client implementation. In this example the default receive buffer size is set to 10000 Bytes. This default value is applied for each new connection as long as the variable `xModifyReceiveBufferSize` is TRUE.

```
IF NOT xInitDone THEN
   // Enable the use of user defined default socket options
   TCPUDP.GVL.G_stDefaultSocketOptionsTCPClient.xModifyReceiveBufferSize := TRUE;
```

```
   // Set the values for the default socket options
   TCPUDP.GVL.G_stDefaultSocketOptionsTCPClient.udiReceiveBufferSizeVal
ue := 10000;
   xInitDone := TRUE;
END_IF
// Connect to the server
IF (fbTcpClient.State = TCPUDP.ET_State.Idle) AND xConnect THEN
   xConnectDone := fbTcpClient.Connect(i_sServerIP:= '192.168.100.11',
i_uiServerPort:= 12345);
   IF NOT xConnectDone THEN
   ;    // error handling
   END_IF
   xConnect :=FALSE;
END_IF
```

NOTE: Independent to the default socket options, it is possible to modify the socket options individually for each open socket. The socket options can be modified with the use of the corresponding **Property** of the respective function block instance.

Part IV
Program Organization Units (POU)

What Is in This Part?

This part contains the following chapters:

Chapter 7
Function Blocks

What Is in This Chapter?

This chapter contains the following sections:

Section 7.1
TCP

What Is in This Section?

This section contains the following topics:

TCP Communication

Overview

TCP connections are used for reliable, stream-based data transfer between systems. One system acts as a server, listening for incoming connections on a specified TCP port and accepting them. The other systems act as clients, connecting to the server on the specified TCP port.

After a connection has been established, data can be transferred in both directions while the TCP protocol helps to ensure that the packets will be delivered in order and that lost packets will be detected and re-sent accordingly.

FB_TCPClient

Overview

Type:	Function block
Available as of:	V1.0.4.0
Inherits from:	-
Implements:	-

Task

Connect to a TCP server to allow data exchange.

Functional Description

The usual order of command is to call the `Connect` method first, specifying the server IP and TCP port to connect to. Afterwards, retrieve the value of the `State` property cyclically until it is different than `Connecting`. If the state is not `Connected`, the connection could not be established. Verify the value of the `Result` property to determine the reason. The method `Close` has to be called before another connection attempt can be started.

Once the state is `Connected`, and the value of the `IsWritable` has changed to TRUE, data can be exchanged using the `Send` and `Receive` methods.

To verify whether there is data ready to be read, the properties `IsReadable` and `BytesAvailableToRead` can be used. Note that if the connection has been closed by the remote site (indicated by the value of the property `PeerHasDisconnected`), `IsReadable` is TRUE even if no bytes can actually be read.

The method `Peek` can be used in the same way as the method `Receive`. The difference is that a call to `Peek` leaves the data in the receive buffer of the TCP, and the data can be read multiple times. This can be used to determine whether enough data has arrived for processing when the length cannot be determined prior to actually having the data available and as such the value of `BytesAvailableToRead` is not sufficient for processing. If enough data has arrived to be properly processed, use, the `Receive` method to remove the data from the receive buffer so that there is space available for more incoming data.

To detect disconnection of the remote site, read the value of the `PeerHasDisconnected` property. The value of `State` automatically changes to `Shutdown` if the connection has been closed by the server. If that case occurs, remaining unprocessed data can still be read but data can no longer be sent.

To close a connection properly, call the `Shutdown` method. The state of the connection changes to `Shutdown` which allows incoming data to be read but no further data can be sent. When the incoming data has been read, processed, or is not of interest, the `Close` method can be called, terminating the connection.

If processing in a method is unsuccessful, it is indicated in the value of the `Result` property. The value of `Result` must be verified after every method call. The result can be reset to `Ok` using the `ResetResult` method.

NOTE: All methods are blocked as long as the value of the property `Result` is unequal to `Ok`. A method call in this case is aborted without affecting the information of the `Result` property.

Interface

The function block does not have inputs and outputs. The functionality is available via methods and properties. You do not need to call the function block directly in your application.

Performance Considerations

When a certain amount of data needs to be available to be processed correctly, two approaches by the application are possible:

- The `Peek` method is used to copy the available data to an empty buffer provided by the application. This data can then be used to evaluate if there is enough data in the buffer. If so, the `Receive` method is used to move the data from the receive buffer of the system to the application and process it there.
- Only the `Receive` method is called, but always on the same buffer provided by the application and specifying the fill level of that buffer. This way, the data is only copied once and the application can evaluate whether there is enough data and process it in one call of the method since the data is already in the buffer used by the application.

The second approach is preferred since it avoids copying data in memory multiple times.

Methods of FB_TCPClient

Overview

FB_TCPClient - Method Close

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Close the socket.

Functional Description

Closes the socket, possibly discarding data in the receive buffer.

The return value is TRUE if the function was executed successfully. Evaluate the property `Result`, in case the return value is FALSE.

State Transition of the Client

Stage	Description
1	Initial state: `Connected`
2	Function call
3	State: `Idle`

FB_TCPClient - Method Connect

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Establish a connection to a TCP server.

Functional Description

Establishes a connection to a TCP server.

The return value is TRUE if the function was executed successfully. Evaluate the property `Result`, in case the return value is FALSE.

NOTE: The return value of this function indicates only whether the connection could be initiated succesfully. The status of the connection must be verified using the property **State**.

State Transition of the Client

Stage	Description
1	Initial state: `Idle`
2	Function call
3	State: `Connecting`

Interface

Input	Data type	Valid range	Description
`i_sServerIP`	STRING(15)	-	IP address of the server to connect to.
`i_uiServerPort`	UINT	1 ... 65535	TCP port of the server to connect to.

FB_TCPClient - Method GetBoundIPAddress

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Return the bound IP address.

Functional Description

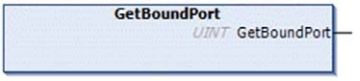
This function is used to obtain the IP address the socket is bound to. If the return value is an empty string (''), the IP address could not be obtained.

This function is supported on platforms where SysSocket library version 3.5.6.0 or later is installed.

FB_TCPClient - Method GetBoundPort

Overview

Type:	Method
Available as of:	V1.0.4.0



Task

Return the bound port

Functional Description

This function is used to obtain the port the socket is bound to. If the return value is 0, the port number could not be obtained.

This function is supported on platforms where SysSocket library version 3.5.6.0 or later is installed.

FB_TCPClient - Method Peek

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Read data stored in the receive buffer without removing it.

Functional Description

Reads data stored in the receive buffer without removing it from there after it has been read.

The Peek method can be used if a certain amount of data needs to be available to be processed correctly and the amount can be determined from parts of the data (a length field for example). In that case, the data can be copied to the application in one call of the `Receive` method.

Returns the number of bytes written to the application-provided buffer as UDINT.

Interface

Input	Data type	Valid range	Description
`i_pbyReceiveBuffer`	POINTER BY BYTE	-	Start address of the buffer to write the received data to.
`i_udiReceiveBufferSize`	UDINT	1 ... 2147483647	Maximum number of bytes the buffer can store.

NOTE: To prevent access violation eventually caused by pointer access to the memory, use the arithmetic operator SIZEOF in conjunction with the targeted buffer to determine the value for `i_udiReceiveBufferSize`.

FB_TCPClient - Method Receive

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Read data stored in the receive buffer and remove it.

Functional Description

Reads data stored in the receive buffer and removes it from there if it has been read without detecting an error. Returns the number of bytes written to the application-provided buffer as UDINT.

For additional information about the receive methods, refer to section Receive Method *(see page 61)*.

Interface

Input	Data type	Valid range	Description
`i_pbyReceiveBuffer`	POINTER TO BYTE	-	Start address of the buffer to write the received data to.
`i_udiReceiveBufferSize`	UDINT	1 ... 2147483647	Maximum number of bytes the buffer can store.

NOTE: To prevent access violation eventually caused by pointer access to the memory, use the arithmetic operator SIZEOF in conjunction with the targeted buffer to determine the value for `i_udiReceiveBufferSize`.

In_Out	Data type	Valid range	Description
iq_udiFillLevel	UDINT	1 ... 2147483647	Fill level of the application-supplied buffer before the operation (data will be written at this offset) and fill level after the received bytes have been written to be buffer.

Receive Methods

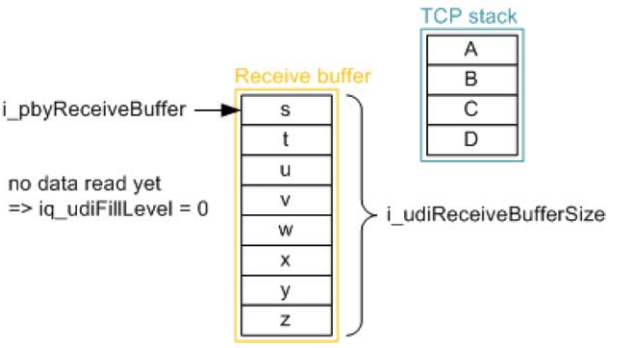
The methods for receiving of data, provided by the function blocks FB_TCPClient and FB_TCPServer in this library provide the input/output parameter iq_udiFillLevel. This parameter determines the offset in the buffer, and therefore where the data is written. On each execution of the function the value is updated by adding the number of the written bytes to the original value.

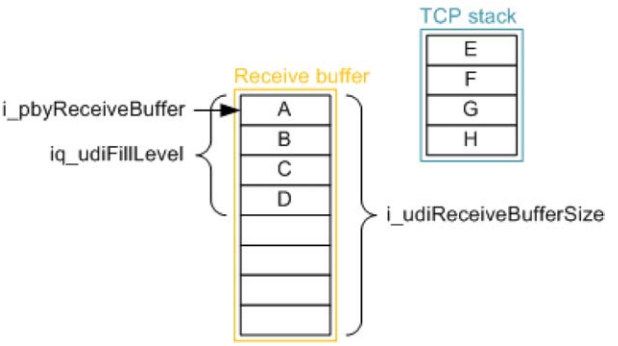
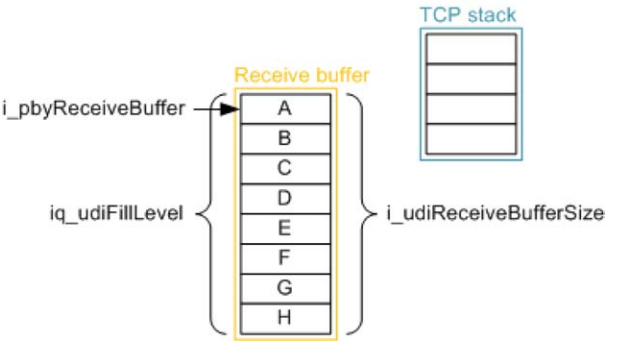
In cases where data are received in several packets, but have to be stored in one buffer and will be processed later as a whole, the respective receive function can be called again and again without modifying the parameter iq_udiFillLevel from the last function call.

The difference of the receive-buffer size (i_udiReceiveBufferSize) and the fill level is used to determine the maximum number of bytes to be read.

Function Call Example

The following graphics illustrate the content of the buffer and the change of parameter iq_udiFillLevel for two function calls, whereby the function was executed successfully each time.

Stage	Description	Illustration
1	Before the first call of the function, the pointer is set to the first index of the buffer. The fill level is set to 0. The parameter i_udiReceiveBufferSize indicates the absolute size of the buffer in bytes.	

Stage	Description	Illustration
2	On each function call, the buffer is erased from the start of the fill level. During the first function call in this example, the available data was moved from the TCP stack into the buffer. The fill level is updated by the function and indicates the number of read bytes in the buffer. When the TCP stack has memory space available, the remote site is informed about the available space, as a result, it sends the subsequent data packet. The second function call is executed without any modification of the input parameters.	
3	During the second function call, the available data is moved again from the TCP stack into the buffer. The fill level is updated by the function and then, the value is equal to the value of `i_udiReceiveBufferSize`. This means that the receive-buffer is full. A further function call would be aborted with the result `FillLevelOutOfRange`. Finally, if the receive-buffer is full, you need to process the data and to update the value of the fill level for the buffer accordingly.	

Data Limits per Function Call

Depending on the controller, the amount of data to be moved in one function call of one of the `Receive` or `Send` methods is limited.

Controller	Number of bytes which can be moved at once*
M258, LMC058, M241, M251	2048 bytes
LMC078	>10.000.000 bytes (limited by the application memory)
*This is the maximum value for the difference between buffer size and fill level.	

FB_TCPClient - Method ReceiveOutOfBand

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Read `OutOfBand` data stored in the receive buffer and remove it.

Functional Description

Reads `OutOfBand` data stored in the receive buffer and removes it from there if it has been read without detecting an error. Returns the number of bytes written to the application-provided buffer as UDINT.

For additional information about the receive methods, refer to section Receive Method *(see page 61)*.

Interface

Input	Data type	Valid range	Description
`i_pbyReceiveBuffer`	POINTER TO BYTE	-	Start address of the buffer to write the received data to.
`i_udiReceiveBufferSize`	UDINT	1 ... 2147483647	Maximum number of bytes the buffer can store.

NOTE: To prevent access violation eventually caused by pointer access to the memory, use the arithmetic operator SIZEOF in conjunction with the targeted buffer to determine the value for `i_udiReceiveBufferSize`.

In_Out	Data type	Valid range	Description
iq_udiFillLevel	UDINT	1 ... 2147483647	Fill level of the application-supplied buffer before the operation (data will be written at this offset) and fill level after the received bytes have been written to be buffer

FB_TCPClient - Method ResetByteCounters

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Reset the counters of total received and sent bytes to 0.

Functional Description

Resets the counters of total received and sent bytes to 0. No return value.

FB_TCPClient - Method ResetResult

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Reset the values of the property `Result` to `Ok`.

Functional Description

Resets the values of the property `Result` to `Ok`. No return value.

FB_TCPClient - Method Send

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Transmits data to the peer.

Functional Description

Transmits data to the peer. The data is read from a buffer supplied by the application. Returns the number of bytes sent to the remote site as UDINT.

For additional information about the send methods, refer to section Send Method *(see page 68)*.

Interface

Input	Data type	Valid range	Description
i_pbySendBuffer	POINTER TO BYTE	-	Start address of the buffer that holds the data to be sent.

In_Out	Data type	Valid range	Description
iq_udiFillLevel	UDINT	1 ... 2147483647	Fill level of the application-supplied buffer before the operation. Set this to the amount of data to send. Since the amount of data that has been sent correctly will be subtracted (and removed from the application-provided buffer), it will be > 0 after the operation if not all data could be sent.

Send Methods

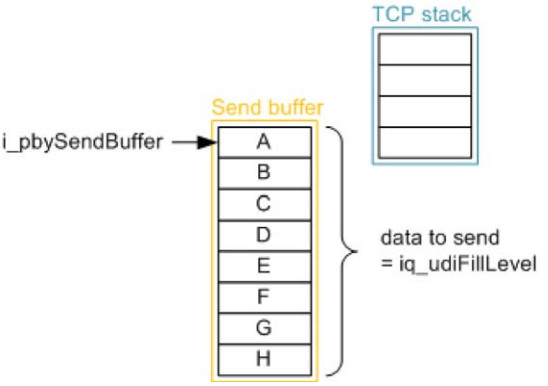
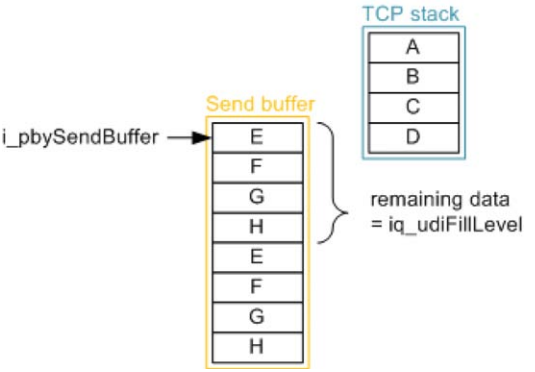
The methods for sending of data, provided by the function blocks FB_TCPClient and FB_TCPServer in this library provide the input/output parameter iq_udiFillLevel. This parameter determines the number of bytes in the buffer which are not yet have been sent. On each execution of the function the value is updated by reducing the number of the sent bytes from the original value. In addition the bytes left in the buffer are copied to the top area of the buffer (data are sent as of start address i_pbySendBuffer).

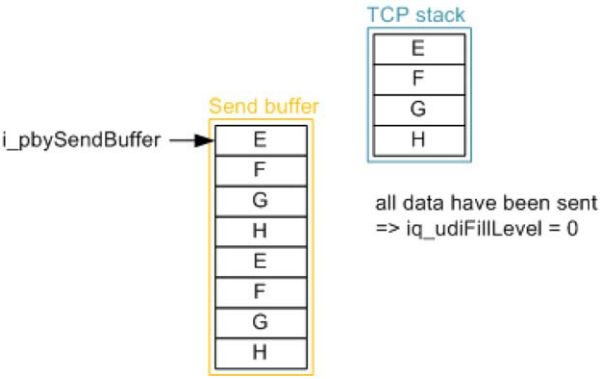
If the fill level is 0 after the function call, all data have been sent and the content of the buffer stays unchanged.

In cases where data could not be copied completely into the TCP stack of the controller in one function call, the respective send function can be called again and again without modifying the parameter iq_udiFillLevel from the last function call and without the need of moving data within the buffer.

Function Call Example

The following graphics illustrate the content of the buffer and the change of the parameter `iq_udiFillLevel` for two function calls, whereby the function was executed successfully each time.

Stage	Description	Illustration
1	Before the first call of the function, the pointer is set to the first index of the buffer. The fill level is set to the number of bytes to be sent. In this illustrated example, the buffer of the TCP stack is empty and its size is less than the send buffer of the application.	
2	During the first function call, the maximum amount of data (size of the TCP stack) has been copied from the send buffer of the application to the TCP stack. The data left in the send buffer of the application have been copied by the function to the top area of the buffer. The parameter `iq_udiFillLevel` has been updated by the function and indicates the number of bytes which could not be sent. The second function call is executed without any modification of the parameters. In the meantime, the TCP stack has sent the data to the remote site so that there is space available again in the TCP stack buffer.	

Stage	Description	Illustration
3	During the second function call, the data left in the send buffer of the application have been copied to the TCP stack. The parameter `iq_udiFillLevel` has been updated by the function and indicates 0. The content of the send buffer is unchanged. A further function call would be aborted with the result `FillLevelOutOfRange`.	

Even though the function supports you in sending data in several function calls, you have to take care of a balanced ratio between:

- Send buffer of the application and send buffer of the TCP socket
- Send buffer local and receive buffer of the remote site
- Send interval of the application and processing time of the remote site

To modify the send buffer size, use the corresponding properties of the function block or adjust the default settings through the global variables in the TCPUDP.GVL (refer to GVL *(see page 45)*).

Data Limits per Function Call

Depending on the controller, the amount of data to be copied in one function call of one of the `Receive` or `Send` method is limited.

Controller	Number of bytes which can be copied at once*
M258, LMC058, M241, M251	2048 bytes
LMC078	>10.000.000 bytes (limited by the application memory)
*This is the maximum value for the difference between buffer size and fill level.	

Special Case - No Data Sent

If the return value of the method indicates 0, no data have been sent and the result of the associated function block is different than `Ok`. Therefore, verify the result with the use of the method `Result` of the function block instance after each function call. If the result indicates `BufferFull`, you must reset the result and try to send the data again during the next program cycle as it is intended for the event if not all data have been sent.

If the result `BufferFull` still appears, optimize the application parameter:

- Increase the send buffer size of the socket
- Increase the receive buffer size of the socket on the remote site
- Adapt the send cycle to the processing time of the remote site

FB_TCPClient - Method SendOutOfBand

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Transmit data as `OutOfBand` data to the peer.

Functional Description

Transmits data as `OutOfBand` data to the peer. The data is read from a buffer supplied by the application. Returns the number of bytes sent to the remote site as UDINT.

For additional information about the send methods, refer to section Send Method *(see page 68)*.

Interface

Input	Data type	Valid range	Description
`i_pbySendBuffer`	POINTER TO BYTE	-	Start address of the buffer that holds the data to be sent.

In_Out	Data type	Valid range	Description
`iq_udiFillLevel`	UDINT	1	Fill level of the application-supplied buffer before the operation. Set this to 1. It will be still 1 after the operation if not all data could be sent.

FB_TCPClient - Method Shutdown

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Send a connection termination to the server.

Functional Description

Sends a connection termination to the server. Afterwards, incoming data can still be read but no further data can be sent. When the incoming data has been received and processed, call the `Close` method.

The return value is TRUE if the function was executed successfully. Evaluate the property `Result`, in case the return value is FALSE.

Properties of FB_TCPClient

Overview

Name	Data type	Access	Description
BytesAvailableToRead	UDINT	Read	Indicates the number of bytes in the receive buffer available to be read using the Receive method. (Range: 0 ... 2147483647)
IsReadable	BOOL	Read	Indicates that data has been received that has not yet been processed by the Receive method.
IsWritable	BOOL	Read	Indicates that the connection is in a state where data can be sent to the server.
PeerHasDisconnected	BOOL	Read	Indicates that the connection has been closed by the remote site. If so, the method Shutdown is automatically called and the state changes to Shutdown.
Result	ET_Result	Read	Indicates the result of the last method call. If the result is different than Ok, the value is not overwritten not even a method is called.
State	ET_State	Read	Indicates the state of the socket.
TotalBytesReceived	ULINT	Read	Counts the total number of bytes received. (Range: 1 ... 2^{64}-1)
TotalBytesSent	ULINT	Read	Counts the total number of bytes sent. (Range: 1 ... 2^{64}-1)
SockOpt_KeepAlive	BOOL	Read/write	If TRUE, instructs the TCP stack to send empty packets periodically to verify whether the remote site is reachable. If this is no longer the case, the connection state changes to Shutdown. **NOTE:** In most cases, set this option so that if the remote site is disconnected (powered off or cable unplugged), this is detected. **NOTE:** If the KeepAlive socket option is disabled for the server, it cannot be enabled for the connected clients.

Name	Data type	Access	Description
SockOpt_NoDelay	BOOL	Read/write	If TRUE, instructs the TCP stack to send the data without waiting for a complete packet. This option might reduce throughput but improves latency which is important in industrial applications.
SockOpt_OutOfBandInline	BOOL	Read/write	If TRUE, instructs the TCP stack to send OutOfBand-data as part of the regular data stream.
SockOpt_ReceiveBufferSize	UDINT	Read/write	Is used to set or get the receive buffer size of the stack. It should always be greater than the amount of data received at one time to help avoid data loss. (Range: 1 ... 2147483647)
SockOpt_SendBufferSize	UDINT	Read/write	Is used to set or get the send buffer size of the stack. It should always be greater than the amount of data sent at one time. (Range: 1 ... 2147483647)

FB_TCPServer

Overview

Type:	Function block
Available as of:	V1.0.4.0
Inherits from:	-
Implements:	-

Task

The TCP server listens for and processes incoming client-connections on a specified port. After a connection has been accepted, data can be received from clients and data can be sent to one or to all clients.

Functional Description

The usual order of commands is to call the `Open` method first, specifying a TCP port number and optionally an IP address of an interface to listen on. When successful, the server is ready to accept incoming connections. It is indicated via the property `IsNewConnectionAvailable`. It should be verified periodically by the application and, in case the value is TRUE, the `Accept` method should be called. It returns the IP address and the port the connection is coming from. You can then programmatically determine whether to maintain the connection. The number of connected clients can be verified using the `NumberOfConnectedClients` property.

To verify whether a client has sent data to the server that is now available to read, use the properties `BytesAvailableToReadFirstAvailableClient` and `BytesAvailableToReadTotal`. While the latter is the sum of the data available from the clients, the former only returns the number of bytes to be read from the first client that has data available. The methods `ReceiveFromFirstAvailableClient` and `PeekFromFirstAvailableClient` can be used to read the data from that client without having the IP and port of the client. Since it is undetermined by the application from which client the data is read from before calling the method, IP and port of the client are provided as outputs of the methods. Unless otherwise noted in the present description, the `FB_TCPServer` methods operate as does the `FB_TCPClient` *(see page 53)*.

The method SendToSpecificClient can be used to reply directly to a specific client after data has been received using the ReceiveFromFirstAvailableClient method. Therefore the application has to provide the IP address and port of a client connected to the TCP server. It then behaves like the Send method of FB_TCPClient.

To send data to all connected clients, use the SendToAll method. When using the SendToAll method, an error from the client terminates the transmission to that client, and the number of bytes sent is returned. Thereafter, you can determine whether all bytes have been sent to all clients by comparing the sum of the bytes sent with the amount of data to be sent multiplied by the number of clients.

While the state of the TCP server state is Listening, the function block instance or the method CheckClients must be called cyclically to detect if a client has closed the connection. Alternatively, the property NumberOfConnectedClients can be retrieved. If a client-initiated disconnection has been detected and there is no more data to be read from that client connection, it is closed and made available for a new incoming connection. Otherwise, the connection is kept available until all data has been read or until the DisconnectClient method is called for that connection, discarding the data that has not yet been processed.

If processing in a method is unsuccessful, it is indicated in the value of the Result property. The value of Result must be verified after every method call. The result can be reset to Ok using the ResetResult method.

NOTE: All methods are blocked as long as the value of the property Result is unequal to Ok. A method call in this case is aborted without affecting the information of the Result property.

Interface

The function block does not have inputs and outputs. The functionality is available via methods and properties. However, it should be called in every cycle while in state Listening so it can detect disconnections from connected clients. Alternatively, the method CheckClients can be called cyclically or the value of the property NumberOfConnectedClients can be retrieved. The three variants verify that clients are still connected or have dropped the connection.

Methods of FB_TCPServer

Overview

FB_TCPServer - Method Accept

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Accept an incoming connection making it available for data transfer.

Functional Description

Accepts an incoming connection making it available for data transfer. The sources IP address and source port the connection is coming from is available as outputs.

The return value is TRUE if the function was executed successfully. Evaluate the property `Result`, in case the return value is FALSE.

Backlog Management

Incoming connections are accepted immediately by the TCP stack and are hold in the backlog. For the client this is a normal connection and it is able to send data to the server. Therefore, it is possible that a connection will be accepted which was already closed again by the client. The data received from this client are still available and as long as the data were not read out using one of the receive methods, the connection stays registered within the property `ConnectedClients`. When the data of such a client connection has been read out using a `Receive` method, this client connection disappears from the list provided with the property `ConnectedClients`.

The number of connections held by the backlog can be set in the GPL of this library with the parameter `Gc_uiTCPServerMaxBacklog` (refer to Global Variables *(see page 41)*).

Interface

Output	Data type	Valid range	Description
q_sClientIP	STRING(15)	-	The IP of the accepted client encoded a string value.
q_dwClientIP	DWORD	-	IP address of the client as DWORD; each byte represents one digit of the IPv4 address.
q_uiClientPort	UINT	1 ... 65535	The source port the client is connecting from.

FB_TCPServer - Method CheckClients

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Verify the client connections for client-initiated disconnections.

Functional Description

Verifies the client connections for client-initiated disconnections and closes the connection if there is no more unread data from the disconnected client. No return value.

FB_TCPServer - Method Close

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Disconnect the clients and close the server socket.

Functional Description

Disconnects the clients and closes the server socket, possibly discarding data in the receive buffers of the client. The server will not be listening for new connections afterwards.

The return value is TRUE if the function was executed successfully. Evaluate the property `Result`, in case the return value is FALSE.

State Transition of the Client

Stage	Description
1	Initial state: `Listening`
2	Functional call
3	State: `Idle`

FB_TCPServer - Method DisconnectAll

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Disconnect the connected clients.

Functional Description

Disconnects the connected clients and keeps listening for new incoming connections.

The return value is TRUE if the function was executed successfully. Evaluate the property `Result`, in case the return value is FALSE.

FB_TCPServer - Method DisconnectClient

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Disconnect a specific client.

Functional Description

Disconnects a specific client.

The return value is TRUE if the function was executed successfully. Evaluate the property `Result`, in case the return value is FALSE.

Interface

Input	Data type	Valid range	Description
`i_sClientIP`	STRING(15)	-	IP address of the client to be disconnected.
`i_uiClientPort`	UINT	1 ... 65535	Source port of the client to be disconnected.

FB_TCPServer - Method GetBoundIPAddress

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Return the bound IP address.

Functional Description

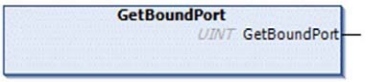
This function is used to obtain the IP address the socket is bound to. If the return value is an empty String (' '), the IP address could not be obtained.

This function is supported on platforms where SysSocket library version 3.5.6.0 or later is installed.

FB_TCPServer - Method GetBoundPort

Overview

Type:	Method
Available as of:	V1.0.4.0



Task

Return the bound port.

Functional Description

This function is used to obtain the port the socket is bound to. If the return value is 0, the port number could not be obtained.

This function is supported on platforms where SysSocket library version 3.5.6.0 or later is installed.

FB_TCPServer - Method Open

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Open the server socket.

Functional Description

Opens the server socket and starts listening for incoming connections.

The return value is TRUE if the function was executed successfully. Evaluate the property `Result`, in case the return value is FALSE.

State Transition of the Server

Stage	Description
1	Initial state: `Idle`
2	Function call
3	State: `Listening`

Interface

Input	Data type	Valid range	Description
`i_sInterfaceIP`	STRING(15)	-	IP address of the interface to bind to. If empty or 0.0.0.0, the server is available on all interfaces.
`i_uiServerPort`	UINT	1 ... 65535	TCP port to listen on.

FB_TCPServer - Method PeekFromFirstAvailableClient

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Read data stored in the receive buffer of the first client that has data available to be read.

Functional Description

Reads data stored in the receive buffer of the first client that has data available to be read without removing it from there after it has been read.

The Peek method can be used if a certain amount of data needs to be available to be processed correctly and the amount can be determined from parts of the data (a length field for example). In that case, the data can be copied to the application in one call of the `Receive` method.

Returns the number of bytes written to the application-provided buffer as UDINT.

Interface

Input	Data type	Valid range	Description
`i_pbyReceiveBuffer`	POINTER TO BYTE	-	Start address of the buffer to write the received data to.
`i_udiReceiveBufferSize`	UDINT	1 ... 2147483647	Maximum number of bytes the buffer can store.

NOTE: To prevent access violation eventually caused by pointer access to the memory, use the arithmetic operator SIZEOF in conjunction with the targeted buffer to determine the value for `i_udiReceiveBufferSize`.

Output	Data type	Valid range	Description
q_sClientIP	STRING(15)	-	IP of the client the data is coming from.
q_dwClientIP	DWORD	-	IP address of the client as DWORD; each byte represents one digit of the IPv4 address.
q_uiClientPort	UINT	-	Source port the data is coming from.

FB_TCPServer - Method PeekFromSpecificClient

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Read data stored in the receive buffer of the client specified by its source IP and port.

Functional Description

Reads data stored in the receive buffer of the client specified by its source IP and port without removing it from there after it has been read.

The Peek method can be used if a certain amount of data needs to be available to be processed correctly and the amount can be determined from parts of the data (a length field for example). In that case, the data can be copied to the application in one call of the `Receive` method.

Returns the number of bytes written to the application-provided buffer as UDINT.

Interface

Input	Data type	Valid range	Description
`i_sClientIP`	STRING(15)	-	IP address of the connected client the data is to be read from.
`i_uiClientPort`	UINT	1 ... 65535	Source port of the connected client the data is to be read from.
`i_pbyReceiveBuffer`	POINTER TO BYTE	-	Start address of the buffer to write the received data to.
`i_udiReceiveBufferSize`	UDINT	1 ... 2147483647	Maximum number of bytes the buffer can store.

FB_TCPServer - Method ReceiveFromFirstAvailableClient

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Read data stored in the receive buffer of the first client that has data available to be read and remove it from there if it has been read without detecting an error.

Functional Description

Reads data stored in the receive buffer of the first client that has data available to be read and removes it from there if it has been read without detecting an error. Returns the number of bytes written to the application-provided buffer as UDINT.

For additional information about the receive methods, refer to section Receive Method *(see page 61)*.

Interface

Input	Data type	Valid range	Description
`i_pbyReceiveBuffer`	POINTER TO BYTE	-	Start address of the buffer to write the received data to.
`i_udiReceiveBufferSize`	UDINT	1 ... 2147483647	Maximum number of bytes the buffer can store.

NOTE: To prevent access violation eventually caused by pointer access to the memory, use the arithmetic operator SIZEOF in conjunction with the targeted buffer to determine the value for `i_udiReceiveBufferSize`.

In_Out	Data type	Valid range	Description
iq_udiFillLevel	UDINT	1 ... 2147483647	Fill level of the application-supplied buffer before the operation (data will be written at this offset) and fill level after the received bytes have been written to be buffer.

Output	Data type	Valid range	Description
q_sClientIP	STRING(15)	-	IP of the client the data is coming from.
q_dwClientIP	DWORD	-	IP address of the client as DWORD; each byte represents one digit of the IPv4 address.
q_uiClientPort	UINT	-	Source port the data is coming from.

FB_TCPServer - Method ReceiveFromSpecificClient

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Read data stored in the receive buffer of the client specified by its source IP and port and removes it from there if it has been read without detecting an error.

Functional Description

Reads data stored in the receive buffer of the client specified by its source IP and port and removes it from there if it has been read without detecting an error. Returns the number of bytes written to the application-provided buffer as UDINT.

For additional information about the receive methods, refer to section Receive Method *(see page 61)*.

Interface

Input	Data type	Valid range	Description
i_sClientIP	STRING(15)	-	IP address of the connected client the data is to be read from.
i_uiClientPort	UINT	1 ... 65535	Source port of the connected client the data is to be read from.
i_pbyReceiveBuffer	POINTER TO BYTE	-	Start address of the buffer to write the received data to.
i_udiReceiveBufferSize	UDINT	1 ... 2147483647	Maximum number of bytes the buffer can store.

NOTE: To prevent access violation eventually caused by pointer access to the memory, use the arithmetic operator SIZEOF in conjunction with the targeted buffer to determine the value for `i_udiReceiveBufferSize`.

In_Out	Data type	Valid range	Description
`iq_udiFillLevel`	UDINT	1 ... 2147483647	Fill level of the application-supplied buffer before the operation (data will be written at this offset) and fill level after the received bytes have been written to be buffer.

FB_TCPServer - Method ReceiveOutOfBandFromFirstAvailableClient

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Read `OutOfBand` data stored in the receive buffer of the first client that has data available to be read.

Functional Description

Reads `OutOfBand` data stored in the receive buffer of the first client that has data available to be read and removes it from there if it has been read without detecting an error. Returns the number of bytes written to the application-provided buffer as UDINT.

For additional information about the receive methods, refer to section Receive Method *(see page 61)*.

Interface

Input	Data type	Valid range	Description
`i_pbyReceiveBuffer`	POINTER TO BYTE	-	Start address of the buffer to write the received data to.
`i_udiReceiveBufferSize`	UDINT	1 ... 2147483647	Maximum number of bytes the buffer can store.

NOTE: To prevent access violation eventually caused by pointer access to the memory, use the arithmetic operator SIZEOF in conjunction with the targeted buffer to determine the value for `i_udiReceiveBufferSize`.

In_Out	Data type	Valid range	Description
iq_udiFillLevel	UDINT	1 ... 2147483647	Fill level of the application-supplied buffer before the operation (data will be written at this offset) and fill level after the received bytes have been written to be buffer.

Output	Data type	Valid range	Description
q_sClientIP	STRING(15)	-	IP of the client the data is coming from.
q_dwClientIP	DWORD	-	IP address of the client as DWORD; each byte represents one digit of the IPv4 address.
q_uiClientPort	UINT	-	Source port the data is coming from.

FB_TCPServer - Method ReceiveOutOfBandFromSpecificClient

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Reads `OutOfBand` data stored in the receive buffer of the client specified by its source IP and port.

Functional Description

Reads `OutOfBand` data stored in the receive buffer of the client specified by its source IP and port and removes it from there if it has been read without detecting an error. Returns the number of bytes written to the application-provided buffer as UDINT.

For additional information about the receive methods, refer to section Receive Method *(see page 61)*.

Interface

Input	Data type	Valid range	Description
`i_sClientIP`	STRING(15)	-	IP address of the connected client the data is to be read from.
`i_uiClientPort`	UINT	1 ... 65535	Source port of the connected client the data is to be read from.
`i_pbyReceiveBuffer`	POINTER TO BYTE	-	Start address of the buffer to write the received data to.
`i_udiReceiveBufferSize`	UDINT	1 ... 2147483647	Maximum number of bytes the buffer can store.

NOTE: To prevent access violation eventually caused by pointer access to the memory, use the arithmetic operator SIZEOF in conjunction with the targeted buffer to determine the value for `i_udiReceiveBufferSize`.

In_Out	Data type	Valid range	Description
iq_udiFillLevel	UDINT	1 ... 2147483647	Fill level of the application-supplied buffer before the operation (data will be written at this offset) and fill level after the received bytes have been written to be buffer.

FB_TCPServer - Method ResetByteCounters

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Reset the counters of total received and sent bytes to 0.

Functional Description

Resets the counters of total received and sent bytes to 0. No return value.

FB_TCPServer - Method ResetResult

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Reset the values of the property `Result` to `Ok`.

Functional Description

Resets the values of the property `Result` to `Ok`. No return value.

FB_TCPServer - Method SendOutOfBandToAll

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Send `OutOfBand` data to the connected clients.

Functional Description

Sends `OutOfBand` data to the connected clients. Errors detected for single clients are ignored. Returns the sum of the sent bytes as UDINT. If this value is equal to the number of connected clients multiplied with the amount of data to be sent, the data was transmitted successfully to the clients.

NOTE: If you need to determine if any individual client had detected an error, then use the `SendOutOfBandToSpecificClient` method. You can retrieve the array of connected clients using the `ConnectedClients` property, and transmit the message to the connected clients individually.

Interface

Input	Data type	Valid range	Description
`i_pbySendBuffer`	POINTER TO BYTE	-	Start address of the buffer that holds the data to be sent.
`i_udiNumBytesToSend`	UDINT	1	Number of bytes in the buffer to be sent. Set this to 1.

FB_TCPServer - Method SendOutOfBandToSpecificClient

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Send `OutOfBand` data only to a specific client, identified by its IP and port.

Functional Description

Sends `OutOfBand` data only to a specific client, identified by its IP and port. Returns the number of bytes sent to the remote site as UDINT.

For additional information about the send methods, refer to section Send Method *(see page 68)*.

Interface

Input	Data type	Valid range	Description
`i_sClientIP`	STRING(15)	-	IP address of the connected client the data is to be sent to.
`i_uiClientPort`	UINT	1 ... 65535	Source port of the connected client the data is to be sent to.
`i_pbySendBuffer`	POINTER TO BYTE	-	Start address of the buffer that holds the data to be sent.

In_Out	Data type	Valid range	Description
iq_udiFillLevel	UDINT	1	Fill level of the application-supplied buffer before the operation. Set this to 1. It will be still 1 after the operation if not all data could be sent.

FB_TCPServer - Method SendToAll

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Send data to the connected clients.

Functional Description

Sends data to the connected clients. Errors detected for single clients are ignored. Returns the sum of the sent bytes as UDINT. If this value is equal to the number of connected clients multiplied with the amount of data to be sent, the data was transmitted successfully to the clients.

NOTE: If you need to determine if any individual client had detected an error, then use the `SendToSpecificClient` method. You can retrieve the array of connected clients using the `ConnectedClients` property, and transmit the message to the connected clients individually.

Interface

Input	Data type	Valid range	Description
`i_pbySendBuffer`	POINTER TO BYTE	-	Start address of the buffer that holds the data to be sent.
`i_udiNumBytesToSend`	UDINT	1 ... 2147483647	Number of bytes in the buffer to be sent.

FB_TCPServer - Method SendToSpecificClient

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Sends data only to a specific client, identified by its IP and port.

Functional Description

Sends data only to a specific client, identified by its IP and port. Returns the number of bytes sent to the remote site as `UDINT`.

For additional information about the send methods, refer to section Send Method *(see page 68)*.

Interface

Input	Data type	Valid range	Description
`i_sClientIP`	STRING(15)	-	IP address of the connected client the data is to be sent to.
`i_uiClientPort`	UINT	1 ... 65535	Source port of the connected client the data is to be sent to.
`i_pbySendBuffer`	POINTER TO BYTE	-	Start address of the buffer that holds the data to be sent.

In_Out	Data type	Valid range	Description
iq_udiFillLevel	UDINT	1 ... 2147483647	Fill level of the application-supplied buffer before the operation. Set this to the amount of data to send. Since the amount of data that has been sent correctly will be subtracted (and removed from the application-provided buffer), it will be greater than 0 after the operation if not all data could be sent.

Properties of FB_TCPServer

Overview

Name	Data type	Access	Description
BytesAvailableToReadFirstAvailableClient	UDINT	Read	Indicates the number of bytes available to be read from the first client that has data available. (Range: 0 ... 2147483647)
BytesAvailableToReadTotal	UDINT	Read	Indicates the total number (sum) of bytes available to be read from the connected clients. (Range: 0 ... 2147483647)
ConnectedClients	ARRAY [1..GPL.Gc_uiTCPServerMaxConnections] OF ST_ClientConnection	Read	Returns an array with information about the connected clients. Refer to ST_ClientConnection *(see page 34)*.
IsDataAvailable	BOOL	Read	Indicates if there is data available to be read from at least 1 client.
IsNewConnectionAvailable	BOOL	Read	Indicates that there is a new incoming connection waiting to be accepted.
NumberOfConnectedClients	UINT	Read	Returns the number of connected clients (including clients that are disconnected but have data to be read). (Range: 0 to GPL.Gc_uiTCPServerMaxConnections)
Result	ET_Result	Read	Indicates the result of the last method call. If the result is different than 0k, the value is not overwritten not even a method is called.
State	ET_State	Read	Indicates the state of the socket.
TotalBytesReceived	ULINT	Read	Counts the total number of bytes received. (Range: 1 ... 2^{64}-1)
TotalBytesSent	ULINT	Read	Counts the total number of bytes sent. (Range: 1 to 2^{64}-1)

Name	Data type	Access	Description
SockOpt_KeepAlive	BOOL	Read/write	If TRUE, instructs the TCP stack to send empty packets periodically to verify whether the remote site is reachable. If this is no longer the case, the connection state changes to Shutdown. **NOTE:** In most cases, set this option so that if the remote site is disconnected (powered off or cable unplugged) this is detected. **NOTE:** If the KeepAlive socket option is disabled for the server, it cannot be enabled for the connected clients.
SockOpt_ReuseAddress	BOOL	Read/write	If TRUE, it allows to open a server even if the port is still bound to, but not actively used by another resource.
SockOpt_ReceiveBufferSize	UDINT	Read/write	Is used to set or get the receive buffer size of the stack. It should always be greater than the amount of data received at one time to help avoid data loss. (Range: 1 ... 2147483647)
SockOpt_SendBufferSize	UDINT	Read/write	Is used to set or get the send buffer size of the stack. It should always be greater than the amount of data sent at 1 time. (Range: 1 ... 2147483647)

Section 7.2
UDP

What Is in This Section?

This section contains the following topics:

UDP Communication

Overview

The UDP protocol is used for connection-less, message-based data exchange between two or more systems, treated as peers. One peer can send a message to another peer (unicast), multiple peers (multicast), or peers on the same subnet (broadcast).

NOTE: The UDP protocol does not set up a dedicated end-to-end connection. The communication between the peers is achieved by transmitting information unidirectional from source to destination. It does not allow you to verify if a message reached its destination peer or information was lost on route. The UDP protocol does not provide any concept of acknowledgment, retransmission, or timeout.

FB_UDPPeer

Overview

Type:	Function block
Available as of:	V1.0.4.0
Inherits from:	-
Implements:	-

Task

Represents an endpoint for sending and receiving messages using the message-based UDP protocol.

Functional Description

The usual order of command is to call the `Open` method first. If this was successful, messages can be sent. If you intend to listen on a specific port, the method `Bind` must be used to bind the socket to this port and optionally to a specific Ethernet interface. If messages are to be received on all available Ethernet interfaces and the outgoing interface shall be used automatically, use an empty string or 0.0.0.0 as the interface-input of the method.

To send data to other peers, use the `Send` method. On the first send from an unbound socket, it is automatically bound and the `Receive` method can be used afterwards. If the runtime supports it, the IP and port the socket got bound to can be requested using the `BoundIPAddress` and `BoundPort` properties.

To verify whether there is data ready to be read, the properties `IsReadable` and `BytesAvailableToRead` can be used.

For both the `Send` and `Receive` methods, a buffer has to be provided by the application that is filled by the `Received` method and contains the data to be sent for the `Send` method.

Broadcasts can be sent and received without any preparation. A multicast group needs to be joined in order to receive multicast messages. Therefore, the methods `JoinMulticastGroup` and `LeaveMulticastGroup` are provided.

If you intend to send UDP-multicast packages using the FB_UDPPeer function block, set the value of the property SockOpt_MulticastDefaultInterface to the IP address of the interface from which the packages should be sent. This has to be performed after calling the Open method and before the first call of the SendTo method.

NOTE: Specifying the default interface for multicast packages by the value of the property SockOpt_MulticastDefaultInterface helps to avoid that the packages are sent to all available network.

The Close method can be used to stop further data transfer and close the socket.

If processing in a method is unsuccessful, it is indicated in the value of the Result property. The value of Result must be verified after every method call. The result can be reset to Ok using the ResetResult method.

NOTE: All methods are blocked as long as the value of the property Result is unequal to Ok. A method call in this case is aborted without affecting the information of the Result property.

Interface

The function block does not have inputs and outputs. The functionality is available via methods and properties. You do not need to call the function block directly in your application.

Methods of FB_UDPPeer

Overview

FB_UDPPeer - Method Bind

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Configure the opened UDP peers for a specified local IP address and port.

Functional Description

Configures the opened UDP peer for a specified local IP address and port as source for messages to be sent and received on.

The return value is TRUE if the function was executed successfully. Evaluate the property `Result`, in case the return value is FALSE.

NOTE: The socket is bound automatically to an available port when data are sent from an unbound socket.

State Transition of the Peer

Stage	Description
1	Initial state: `Opened`
2	Function call
3	State: `Bound`

Interface

Input	Data type	Valid range	Description
`i_sLocalIP`	STRING(15)	-	IP address of the interface to bind on. If empty or 0.0.0.0, the peer listen to all interfaces.
`i_uiLocalPort`	UINT	1 ... 65535	UDP port to bind on.

FB_UDPPeer - Method Close

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Close the socket.

Functional Description

Closes the socket, possibly discarding data in the receive buffer. The multicast-groups joined are automatically left.

The return value is TRUE if the function was executed successfully. Evaluate the property `Result`, in case the return value is FALSE.

State Transition of the Peer

Stage	Description
1	Initial state: `Opened` or `Bound`
2	Function call
3	State: `Idle`

FB_UDPPeer - Method GetBoundIPAddress

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Return the bound IP address.

Functional Description

This function is used to obtain the IP address the socket is bound to. If the return value is an empty string (''), the IP address could not be obtained.

This function is supported on platforms where SysSocket library version 3.5.6.0 or later is installed.

FB_UDPPeer - Method GetBoundPort

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Return the bound port.

Functional Description

This function is used to obtain the port the socket is bound to. If the return value is 0, the port number could not be obtained.

This function is supported on platforms where SysSocket library version 3.5.6.0 or later is installed.

FB_UDPPeer - Method JoinMulticastGroup

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Join a multicast group for receiving messages.

Functional Description

Joins a multicast group for receiving messages sent to that group address by sending an IGMP `AddMembership` message (Internet Group Management Protocol).

The return value is TRUE if the function was executed successfully. Evaluate the property `Result`, in case the return value is FALSE.

Interface

Input	Data type	Valid range	Description
`i_sInterfaceIP`	STRING(15)	-	IP address of the interface to join the multicast group on.
`i_sGroupIP`	STRING(15)	-	Multicast address of the group to be joined.

FB_UDPPeer - Method LeaveMulticastGroup

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Leave a multicast group.

Functional Description

Leaves a multicast group by sending an IGMP `DropMembership` message. After the multicast group has been left, messages sent to that group address will not be received anymore.

The return value is TRUE if the function was executed successfully. Evaluate the property `Result`, in case the return value is FALSE.

Interface

Input	Data type	Valid range	Description
`i_sInterfaceIP`	STRING(15)	-	IP address of the interface to leave the multicast group on.
`i_sGroupIP`	STRING(15)	-	Multicast address of the group to leave.

FB_UDPPeer - Method Open

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Initialize and open the UDP peers.

Functional Description

Initializes and opens the UDP peer.

The return value is TRUE if the function was executed successfully. Evaluate the property `Result`, in case the return value is FALSE.

NOTE: If you intend to listen on a specific port, the method `Bind` must be used subsequently to the method `Open` to bind the open socket.

State Transition of the Peer

Stage	Description
1	Initial state: `Idle`
2	Function call
3	State: `Opened`

FB_UDPPeer - Method ReceiveFrom

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Read data stored in the receive buffer.

Functional Description

Reads data stored in the receive buffer and removes it from the buffer if it has been read without detecting an error. At most one message is read no matter how much data is available to be read and how large is the application-provided buffer. Returns the number of bytes written to the application-provided buffer as UDINT.

Interface

Input	Data type	Valid range	Description
i_pbyReceiveBuffer	POINTER TO BYTE	-	Start address of the buffer to write the received data to.
i_udiReceiveBufferSize	UINT	1 ... 2147483647	Maximum number of bytes the buffer can store.

Output	Data type	Valid range	Description
q_xDataReceived	BOOL	-	Indicates whether a message was received.
q_sPeerIP	STRING(15)	-	Source IP of the peer from whom the message was received in STRING representation.

Output	Data type	Valid range	Description
q_dwPeerIP	DWORD	-	IP address of the peer (sender) as DWORD; each byte represents one digit of the IPv4 address.
q_uiPeerPort	UINT	-	Source port the message was received from.

FB_UDPPeer - Method ResetByteCounters

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Reset the counters of total received and sent bytes to 0.

Functional Description

Resets the counters of total received and sent bytes to 0. No return value.

FB_UDPPeer - Method ResetResult

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Reset the values of the property `Result` to `Ok`.

Functional Description

Resets the values of the property `Result` to `Ok`. No return value.

FB_UDPPeer - Method SendTo

Overview

Type:	Method
Available as of:	V1.0.4.0

Task

Transmit one message.

Functional Description

Transmits one message. The data is read from a buffer supplied by the application. This method is used for sending unicast, multicast, or broadcast messages. If the socket was not bound before, it is automatically bound to an available port. Returns the number of bytes sent as UDINT.

Interface

Input	Data type	Valid range	Description
i_sPeerIP	STRING(15)	-	Destination address to send the message to.
i_uiPeerPort	UINT	-	Destination port to send the message to.
i_pbySendBuffer	POINTER TO BYTE	-	Start address of the buffer that holds the data to be sent.
i_udiNumBytesToSend	UDINT	1 ... 2147483647	Number of bytes in the application-provided buffer that shall be sent.

Properties of FB_UDPPeer

Overview

Name	Data type	Accessing	Description
BytesAvailableToRead	UDINT	Read	Indicates the number of bytes in the receive buffer available to be read using the Receive method. (Range: 0 ... 2147483647)
IsReadable	BOOL	Read	Indicates that data has been received that has not yet been processed by the Receive method.
IsWritable	BOOL	Read	Indicates that the connection is in a state where data can be sent to the server.
Result	ET_Result	Read	Indicates the result of the last method call. If the result is different than Ok, the value is not overwritten not even a method is called.
State	ET_State	Read	Indicates the state of the socket.
TotalBytesReceived	ULINT	Read	Counts the total number of bytes received. (Range: 1 ... 2^{64}-1)
TotalBytesSent	ULINT	Read	Counts the total number of bytes sent. (Range: 1 ... 2^{64}-1)
SockOpt_Broadcast	BOOL	Read/write	Allows you to send broadcast packets via that UDP socket. If FALSE, the Send method returns an error message when sending UDP broadcast messages.
SockOpt_MulticastDefaultInterface	STRING(15)]	Read/write	Allows you to specify the IP address of the interface that is used to send multicast messages when no action has been performed that caused the socket to be bound.
SockOpt_MulticastLoopback	BOOL	Read/write	If set to TRUE, multicast messages sent are also copied to the receive buffer as if they had been sent by an external UDP peer.
SockOpt_MulticastTTL	SINT	Read/write	Specifies the time to live (TTL) of multicast messages sent. This value can affect the scope of where the packages are forwarded to. (Range: 0 ... 255)
SockOpt_ReceiveBufferSize	UDINT	Read/write	Sets the receive buffer size of the UDP stack. It should always be greater than the amount of data received at one time to help avoid data loss. (Range: 1 ... 2147483647)
SockOpt_SendBufferSize	UDINT	Read/write	Sets the send buffer size of the UDP stack. It should always be greater than the amount of data sent at one time to help avoid errors. (Range: 1 ... 2147483647)

Section 7.3
Utils

FB_DnsClient

Overview

Type:	Function block
Available as of:	V1.1.0.0

Task

Communicates with the specified DNS server to request the resolution of a domain name into an IPv4 address.

Functional Description

This function block is used to communicate with a DNS server (according to RFC1035) in order to get the registered IPv4 address corresponding to the specified domain name. Therefore, an UDP socket is open and a DNS request is sent to the server, which is specified by the inputs `i_sDnsServerIp` and `i_uiDnsServerPort`. When a response from the server has been received or the timeout is reached, the socket is closed again.

NOTE: The function block supports authoritative and recursive answers provided by the DNS server.

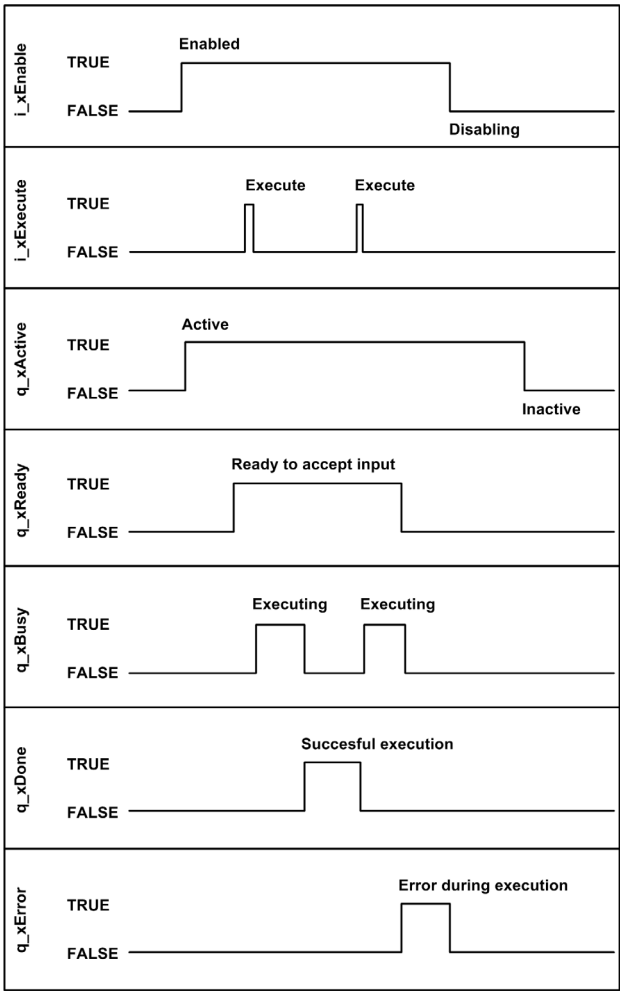
If the domain name could be resolved by the server and the response has been received correctly, the resolved IPv4 addresses and corresponding time to live (TTL) are available on the output `q_astDnsAddressInfo`. To limit the network traffic, the TTL value can be used to cache the resolved addresses.

NOTE: Refresh the IP-Cache according to the information provided by TTL.

The communication with the DNS server takes several program cycles. The status of the function block is indicated by the outputs `q_xBusy`, `q_xError`, and `q_xDone`.

As long as the function block is executed, the output `q_xBusy` is set to TRUE. The output `q_xDone` is set to TRUE after the function block has been executed successfully.

Status messages and diagnostic information are provided using the outputs `q_xError` (TRUE if an error has been detected), `q_etResult`, and `q_etResultMsg`.



To acknowledge detected errors, disable and re-enable the function block to be able to execute a new attempt to resolve the domain name.

Interface

Input	Data type	Description
i_xEnable	BOOL	Activation and initialization of the function block.
i_xExecute	BOOL	Upon a rising edge of this input the DNS request is sent to the DNS server.
i_sDnsServerIP	STRING(15)	Specifies the IP address of the external DNS server.
i_uiDnsServerPort	UINT	Specifies the port of the external DNS server. If the pin is not assigned the default value 53 is used.
i_sDomainName	STRING(255)	Domain name to be resolved.(Only ASCII symbols are supported)

Output	Data type	Description
q_xActive	BOOL	If the function block is active, this output is set to TRUE.
q_xReady	BOOL	Indicates TRUE if the function block is ready to receive an execute command.
q_xBusy	BOOL	If this output is set to TRUE, the function block execution is in progress.
q_xDone	BOOL	If this output is set to TRUE, the execution has been completed successfully.
q_xError	BOOL	If this output is set to TRUE, an error has been detected. For details, refer to q_etResult and q_etResultMsg.
q_etResult	ET_Result	Provides diagnostic and status information as a numeric value.
q_sResultMsg	STRING(80)	Provides additional diagnostic and status information as a text message.
q_uiNumberOfIpAddresses	UINT	Number of IP addresses the DNS server returned.
q_astDnsAddressInfo	ARRAY [0..GPL.Gc_uiDnsNumberOfIPs-1] OF ST_DnsAddressInfo	Structure contains information about the resolved domain name received from the DNS server.

Chapter 8
Functions

What Is in This Chapter?

This chapter contains the following sections:

Section 8.1
Data Types (EnumToStringConverters)

What Is in This Section?

This section contains the following topics:

FC_EtResultToString

Overview

Type:	Function
Available as of:	V1.0.4.0
Inherits from:	–
Implements:	–

Task

Convert an enumeration element of type ET_Result to a variable of type STRING.

Functional Description

Using the function `FC_EtResultToString`, you can convert an enumeration element of type ET_Result to a variable of type STRING.

Interface

Input	Data type	Description
`i_etResult`	ET_Result	Enumeration with the result.

Return Value

Data type	Description
STRING(80)	The ET_Result converted to text.

FC_EtStateToString

Overview

Type:	Function
Available as of:	V1.0.4.0
Inherits from:	-
Implements:	-

Task

Convert an enumeration element of type ET_State to a variable of type STRING.

Functional Description

Using the function `FC_EtStateToString`, you can convert an enumeration element of type ET_State to a variable of type STRING.

Interface

Input	Data type	Description
`i_etState`	ET_State	Enumeration with the present state.

Return Value

Data type	Description
STRING(80)	The ET_State converted to text.

Section 8.2
Utils

What Is in This Section?

This section contains the following topics:

FC_GetSubNetBroadcastAddr

Overview

Type	Function
Available as of	V1.0.4.0
Inherits from	-
Implements	-

Task

Calculate the broadcast address of a subnet.

Functional Description

This function calculates a subnet broadcast IP for a specific IP and a subnet mask. A packet sent to this broadcast IP is received by the devices in this subnet. Use a subnet broadcast IP instead of a broadcast IP (255.255.255.255).

Interface

Input	Data type	Description
`i_sIPInSubnet`	STRING(15)	The IPv4 addresses of the subnet.
`i_sSubnetMask`	STRING(15)	The subnet mask.

Return Value

Data type	Description
STRING(15)	Broadcast IP of the subnet

FC_InetAddrDWORDtoString

Overview

Type	Function
Available as of	V1.0.4.0
Inherits from	-
Implements	-

Task

Convert the IPv4 address given as DWORD into STRING.

Functional Description

This function converts an IP address from a DWORD representation into a STRING representation.

Interface

Input	Data type	Description
i_dwAddress	DWORD	The IPv4 address as DWORD.

Return Value

Data type	Description
STRING(15)	The IPv4 address as STRING.

FC_InetAddrStringtoDWORD

Overview

Type	Function
Available as of	V1.0.4.0
Inherits from	-
Implements	-

Task

Convert the IPv4 address given as STRING into DWORD.

Functional Description

This function converts an IPv4 address from a STRING representation into a DWORD representation.

Interface

Input	Data type	Description
i_sAddress	STRING(15)	The IPv4 address as STRING.

Return Value

Data type	Description
DWORD	The IPv4 address as DWORD.

FC_IsMulticastIP

Overview

Type	Function
Available as of	V1.0.4.0
Inherits from	-
Implements	-

Task

Determine whether a given IPv4 address is in the multicast range.

Functional Description

This function determines whether the given IPv4 address is in the range of multicast addresses (224.0.0.0 to 239.255.255.255) according to RFC5771.

Interface

Input	Data type	Description
i_sIP	STRING(15)	Any IPv4 address

Return Value

Data type	Description
BOOL	TRUE in case of multicast address, FALSE otherwise.

FC_IsValidIP

Overview

Type	Function
Available as of	V1.0.4.0
Inherits from	-
Implements	-

Task

Determine whether the given string is a valid IPv4 address.

Functional Description

This function determines whether the given string is a valid IPv4 address.

Interface

Input	Data type	Description
i_sIP	STRING(15)	String to be verified.

Return Value

Data type	Description
BOOL	TRUE if the given string is a valid IPv4 address, FALSE otherwise.

FC_ReadSTRING

Overview

Type	Function
Available as of	V1.0.4.0
Inherits from	-
Implements	-

Task

Copy characters stored in any data type into a variable of type STRING.

Functional Description

With the use of this function, potentially received ASCII characters can be easily copied from the receive buffer into a variable of type STRING.

The data source of any data type is passed to the function with the use of a pointer through the input `i_pbyBuffer`. The destination for the data, the variable of type STRING, is passed to the function with the use of a pointer through the input `i_psString`. The maximum number of characters to be copied is determined through the input `i_uiMaxLength`.

Through the input `i_xStopAtZero`, it is determined whether the function shall copy all bytes specified by `i_uiMaxLength` or if the copying process shall stop at the first NUL character (16#0) detected. Note that the NUL character indicates the end of the value of a variable of type STRING.

If no NUL character is detected until the maximum number of characters are copied or the input `i_xStopAtZero` is FALSE, the function will write the NUL character into the n^{th} byte of the destination memory. n = `i_uiMaxLength` +1. This is, the maximum value of `i_uiMaxLength` equals to the size of the destination memory - 1.

NOTE: To prevent access violation eventually caused by pointer access to the memory, use the arithmetic operator SIZEOF in conjunction with the destination memory to determine the value for `i_uiMaxLength`.

Coding Example

Coding example for the use of the FC_ReadSTRING in structured text:

```
// copy the data into a variable of type STRING
TCPUDP.FC_ReadSTRING(
     i_pbyBuffer  := ADR(abyReceiveBuffer),
     i_psString  := ADR(sData),
     i_uiMaxLength  := SIZEOF(sData)-1,
     i_xStopAtZero  := TRUE);
```

Interface

Input	Data type	Description
i_pbyBuffer	POINTER TO BYTE	Pointer to memory address to be copied from (source).
i_psString	POINTER TO STRING	Pointer to memory address to be copied to (destination, a variable of type STRING)
i_uiMaxLength	UINT	Maximum number of bytes to be copied.
i_xStopAtZero	BOOL	If TRUE, the copying process stops when the first NUL character (16#0) was detected. If FALSE the number of bytes specified with i_uiMaxLength and one NUL character are written into the destination memory.

Return Value

Data type	Description
UINT	Number of bytes written into the destination memory.

FC_WriteSTRING

Overview

Type	Function
Available as of	V1.0.4.0
Inherits from	-
Implements	-

Task

Copy characters stored from a variable of type STRING into a buffer of any data type.

Functional Description

With the use of this function the value of a variable of type STRING can be easily copied into the send buffer of any data type.

The data source, the variable of type STRING, is passed with the use of a pointer through the input `i_psString`. The destination for the data, a buffer of any data type, is passed to the function with the use of a pointer through the input `i_pbyBuffer`. The maximum number of characters to be copied is determined through the input `i_uiMaxLength`.

If the length of the value of the variable of type STRING is less than the value of `i_uiMaxLength`, the function stops the copying process when the first NUL character (16#0) is detected. Note that the NUL character indicates the end of the value of a variable of type STRING. If the length is greater than the value of `i_uiMaxLength`, the maximum number of bytes is copied and the last byte is overwritten with 16#0 in the buffer.

NOTE: To prevent access violation eventually caused by pointer access to the memory, use the arithmetic operator SIZEOF in conjunction with the targeted buffer to determine the value for `i_uiMaxLength`.

Coding Example

Coding example for the use of the `FC_WriteSTRING` in structured text:

```
// copy the value of the variable of type STRING into the buffer
```

```
TCPUDP.FC_WriteSTRING(
     i_pbyBuffer  := ADR(abySendBuffer),
     i_psString  := ADR(sData),
     i_uiMaxLength  := SIZEOF(abySendBuffer);
```

Interface

Input	Data type	Description
`i_pbyBuffer`	POINTER TO BYTE	Pointer to memory address to be copied to destination memory.
`i_psString`	POINTER TO STRING	Pointer to memory address to be copied from (source variable of type STRING).
`i_uiMaxLength`	UINT	Maximum number of bytes to be copied.

Return Value

Data type	Description
UINT	Number of bytes written into destination memory.

Section 8.3
Utils (Byteorder)

What Is in This Section?

This section contains the following topics:

FC_Read<Data type>

Overview

Type	Function
Available as of	V1.0.4.0
Inherits from	-
Implements	-

Example `FC_ReadDINT`

Task

Read a value from a buffer, convert it to the byte order of the controller, and return it as the special data type.

Functional Description

The function reads the value from a buffer, expected in network byte order, converts it to the byte order of the controller and returns it as the special data type <Data type>.

Functions Available

The following functions are available for the different data types:

Function	Data type
FC_ReadDINT	DINT
FC_ReadINT	INT
FC_ReadSINT	SINT
FC_ReadUDINT	UDINT
FC_ReadUINT	UINT
FC_ReadUSINT	USINT

Interface

Input	Data type	Description
`i_pbyBuffer`	POINTER TO BYTE	Start address of the buffer to read.

Return Value

Data type	Description
<Data type> (see table above)	Value in <Data type>.

FC_Write<Data type>

Overview

Type	Function
Available as of	V1.0.4.0
Inherits from	-
Implements	-

Example FC_WriteDINT

Task

Write the value in the special data type into the expected in network byte order to the buffer.

Functional Description

The function writes the value in the special <Data type> into the expected in network byte order to the buffer and returns TRUE if it is done.

Functions Available

The following functions are available for the different data types:

Function	Data type
FC_WriteDINT	DINT
FC_WriteINT	INT
FC_WriteSINT	SINT
FC_WriteUDINT	UDINT
FC_WriteUINT	UINT
FC_WriteUSINT	USINT

Interface

Input	Data type	Description
`i_pbyBuffer`	POINTER TO BYTE	Start address of the buffer to write the value.
`i_<Data type>Value` (for example i_diValue for `FC_WriteDINT`)	<Data type> (see table above)	Value in the special data type.

Return Value

Data type	Description
BOOL	TRUE if done.

Glossary

A

address

In most cases, the address of a system is the IP address (for example, 192.168.5.45). If you talk about the address of a server, client or peer, this also comprises the port number (for example, 192.168.5.45:5548).

application

A program including configuration data, symbols, and documentation.

B

broadcast

See “unicast”.

BSD

Berkeley Software Distribution

C

client

A client is a part of a communications application. The initially active part establishes a connection (TCP) or sends data to the server.

configuration

The arrangement and interconnection of hardware components within a system and the hardware and software parameters that determine the operating characteristics of the system.

controller

Automates industrial processes (also known as programmable logic controller or programmable controller).

D

DNS

Domain Name System

E

Ethernet

A physical and data link layer technology for LANs, also known as IEE 802.3.

Ethernet is the most widely spread technology for local networks. Each PacDrive controller has an Ethernet port. The Ethernet standard defines layer 1 and 2 of the communication. Above the Ethernet, there are many different network protocols but only IP is used.

expansion bus

An electronic communication bus between expansion I/O modules and a controller.

I

I/O

(*input/output*)

IGMP

The Internet Group Management Protocol (IGMP) is a communications protocol used by hosts and adjacent routers on IPv4 networks to establish multicast group memberships.

IP

IP (Internet Protocol) is the protocol below TCP and UDP, which transports the data across the network.

IP is part of the TCP/IP protocol family that tracks the Internet addresses of devices, routes outgoing messages, and recognizes incoming messages.

IP address

The IP address of IPv4 (Internet Protocol version 4) is a value of 4 bytes which identifies the devices connected to an IP network.

M

multicast

See “unicast”.

P

packet and datagram

On network level, the term packet is used for the data packets which this level transmits. In the case of UDP, these terms are used synonymously.

peer

Term for the other system participating in the communication. This term is used if it is unimportant whether the other system is a server or a client.

port / port number

A port number, frequently also designated as port, is a number from 1 to 65535. In combination with an IP address, it designates a communication end point. A socket is always connected with a port number. As sockets are used by the communication function blocks of TCP/UDP communication, and these again by a program, a port number identifies a program, a server, or a client running on a controller.

If you communicate with the `<IP of a controller>:<Port number>`, then you communicate with a program that has connected itself to this port number. (The program has configured its communication function block such that it connects to this port number.)

program

The component of an application that consists of compiled source code capable of being installed in the memory of a logic controller.

S

server

A server is a part of a communications application. At first, the server is passive. It waits until clients initialize communication. A server runs on a defined port number and the clients know its address.

socket

A socket is a resource which is used by TCP/UDP communication internally in order to access the communications functions in the firmware.

T

TCP

TCP (Transmission Control Protocol) is a transmission protocol used in IP networks.

TTL

TTL (Time to live) or hop limit is a mechanism that limits the lifespan or lifetime of data in a computer or network.

Under the Internet Protocol (IP), TTL can be implemented as a counter or timestamp attached to or embedded in the data. Once the defined event count or timespan has elapsed, data is discarded.

Under DNS, TTL prevents a data packet from circulating indefinitely.

U

UDP

UDP (User Datagram Protocol) is a transmission protocol used in IP networks.

unicast / broadcast /multicast

Unicast communication targets one system as destination. For unicast communication, either TCP or UDP can be used.

Broadcast communication targets all systems in the same subnet. As with multicast, only UDP can be used.

Multicast targets some systems that have to join a specified multicast-group before being delivered packets targeted at that group.

Index

Symbols

E

F

G

M

Q

S

T