

Unity Pro

TCP Open Block Library

04/2015

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.

No part of this document may be reproduced in any form or by any means, electronic or mechanical, including photocopying, without express written permission of Schneider Electric.

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.

© 2015 Schneider Electric. All rights reserved.

Table of Contents



	Safety Information	5
	About the Book	7
Part I	General	9
Chapter 1	Block Types and their Applications	11
	Block Types	12
	FFB Structure	14
	EN and ENO	17
Chapter 2	Block Availability on Various Hardware Platforms ..	21
	Availability of the Blocks on the Various Hardware Platforms	21
Part II	Introduction to TCP Open	23
Chapter 3	General	25
	General Points and Principles of TCP Open	25
Chapter 4	Warnings	27
	Notes and Warnings	27
Chapter 5	Description of Operation	29
	Operating Rules for TCP Open EFs	30
	Principle Governing Socket Operation	31
	General Structure of a TCP Open Communication EF	33
	Structure of TCP Open Management Parameters	34
	Management Parameters: Communication and Operation Reports ..	36
	The Client/Server Model	38
	Example of Client/Server Applications	40
Chapter 6	Operating Modes and Performance	45
	Operating Modes of the Network Module	46
	Performance	48
	Debugging and Diagnostics	49
Part III	Advanced	51
Chapter 7	FCT_ACCEPT: Accepts a Connection Request	53
	Description	53
Chapter 8	FCT_BIND: Binds a Socket Number to an IP Address and a Port	57
	Description	57
Chapter 9	FCT_CLOSE: Deletes the Specified Socket	61
	Description	61

Chapter 10	FCT_CONNECT: Establishes a Connection with an IP Address	65
	Description	65
Chapter 11	FCT_LISTEN: Configuration of a Socket Await Connection	69
	Description	69
Chapter 12	FCT_RECEIVE: Retrieves Data Available on a Socket	71
	Description	71
Chapter 13	FCT_SELECT: Multiplexes Requests Over Sockets ..	75
	Description	75
Chapter 14	FCT_SEND: Sending Data to a Specified Socket	79
	Description	79
Chapter 15	FCT_SETSOCKOPT: Sets the Options Associated with the Socket	83
	Description	83
Chapter 16	FCT_SHUTDOWN: Disables Transmission on the Socket	87
	Description	87
Chapter 17	FCT_SOCKET: Creation of a New Socket	91
	Description	91
Part IV	DFB Library	93
Chapter 18	DFB Library	95
	DFB Library Introduction	96
	TCP_CNX DFB	97
	TCP_SEND DFB	100
	TCP_RECEIVE DFB	103
Glossary		107
Index		109

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, 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.

About the Book



At a Glance

Document Scope

This document describes the functions and function blocks of the TCP Open library.

Validity Note

This document is valid for Unity Pro 10.0 or later.

Part I

General

Overview

This part contains general information about the TCP Open library.

What Is in This Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
1	Block Types and their Applications	11
2	Block Availability on Various Hardware Platforms	21

Chapter 1

Block Types and their Applications

Overview

This chapter describes the different block types and their applications.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Block Types	12
FFB Structure	14
EN and ENO	17

Block Types

Block Types

Different block types are used in Unity Pro. The general term for the block types is FFB.

There are the following types of block:

- Elementary Function (EF)
- Elementary Function Block (EFB)
- Derived Function Block (DFB)
- Procedure

NOTE: Motion Function Blocks are not available on the Quantum platform.

Elementary Function

Elementary functions (EF) have no internal status and one output only. If the input values are the same, the output value is the same for the executions of the function, for example the addition of two values gives the same result at every execution.

An elementary function is represented in the graphical languages (FBD and LD) as a block frame with inputs and an output. The inputs are represented on the left and the outputs on the right of the block frame. The name of the function, that is the function type, is shown in the center of the block frame.

The number of inputs can be increased with some elementary functions.

CAUTION

UNEXPECTED BEHAVIOR OF EQUIPMENT

For Unity Pro V4.0 and earlier versions, do not use links to connect function blocks outputs, when your application relies on persistent output data of an EF.

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

NOTE: With Unity Pro V4.0 and earlier versions the deactivation of an EF (EN=0) causes links connected to its Input/Output to be reset. To transfer the state of the signal do not use a link. A variable must be connected to the EF's output and must be used to connect the input of the element. With Unity Pro V4.1 and later versions you can maintain the output links even if an EF is deactivated by activating the option **Maintain output links on disabled EF (EN=0)** via the menu **Tools →Program →Languages →Common**.

Elementary Function Block

Elementary function blocks (EFB) have an internal status. If the inputs have the same values, the value on the outputs can have another value during the individual executions. For example, with a counter, the value on the output is incremented.

An elementary function block is represented in the graphical languages (FBD and LD) as a block frame with inputs and outputs. The inputs are represented on the left and the outputs on the right of the block frame. The name of the function block, that is the function block type, is shown in the center of the block frame. The instance name is displayed above the block frame.

Derived Function Block

Derived function blocks (DFBs) have the same properties as elementary function blocks. They are created by the user in the programming languages FBD, LD, IL and/or ST.

Procedure

Procedures are functions with several outputs. They have no internal state.

The only difference from elementary functions is that procedures can have more than one output and they support variables of the `VAR_IN_OUT` data type.

Procedures do not return a value.

Procedures are a supplement to IEC 61131-3 and must be enabled explicitly.

There is no visual difference between procedures and elementary functions.

CAUTION

UNEXPECTED BEHAVIOR OF EQUIPMENT

For Unity Pro V4.0 and earlier versions, do not use links to connect function blocks outputs, when your application relies on persistent output data of an EF.

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

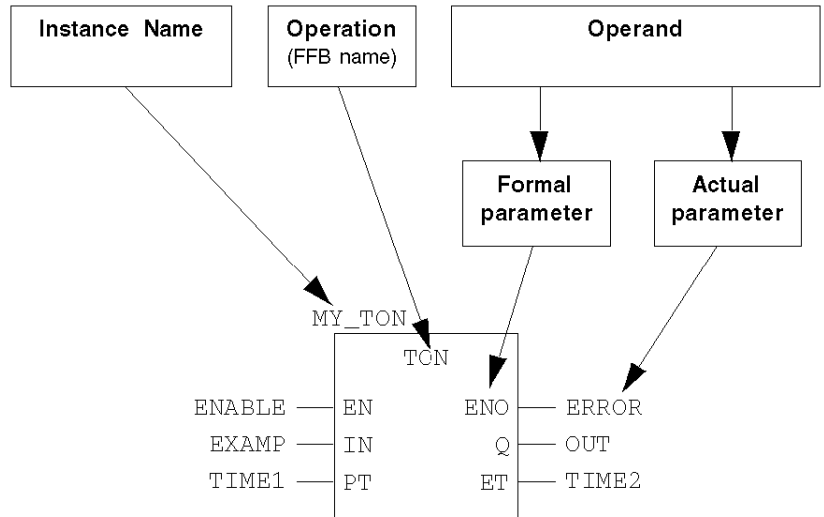
NOTE: With Unity Pro V4.0 and earlier versions the deactivation of an EF (EN=0) causes links connected to its Input/Output to be reset. To transfer the state of the signal do not use a link. A variable must be connected to the EF's output and must be used to connect the input of the element. With Unity Pro V4.1 and later versions you can maintain the output links even if an EF is deactivated by activating the option **Maintain output links on disabled EF (EN=0)** via the menu **Tools → Program → Languages → Common**.

FFB Structure

Structure

Each FFB is made up of an operation (name of the FFB), the operands are required for the operation (formal and actual parameters) and an instance name for elementary/derived function blocks.

Call of a function block in the FBD programming language:



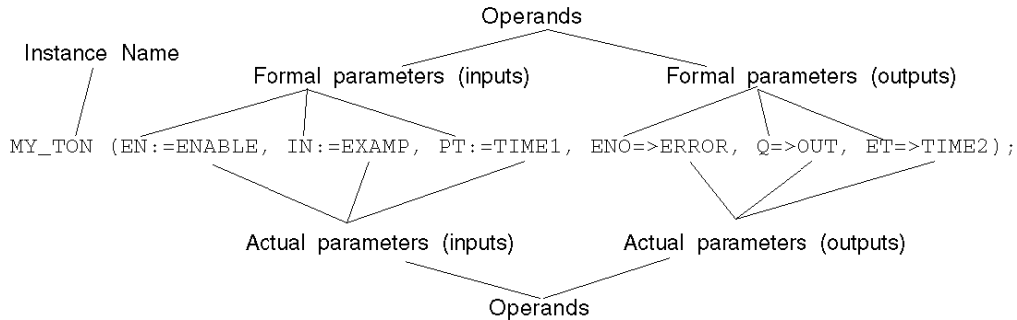
⚠ CAUTION

UNEXPECTED APPLICATION BEHAVIOR

Do not call several times the same block instance within a PLC cycle

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

Formal call of a function block in the ST programming language:



Operation

The operation determines which function is to be executed with the FFB, e.g. shift register, conversion operations.

Operand

The operand specifies what the operation is to be executed with. With FFBs, this consists of formal and actual parameters.

Formal/actual parameters

Inputs and outputs are required to transfer values to or from an FFB. These are called formal parameters.

Objects are linked to formal parameters; these objects contain the current process states. They are called actual parameters.

At program runtime, the values from the process are transferred to the FFB via the actual parameters and then output again after processing.

The data type of the actual parameters must match the data type of the input/output (formal parameters). The only exceptions are generic inputs/outputs whose data type is determined by the actual parameter. If the actual parameters consist of literals, a suitable data type is selected for the function block.

FFB Call in IL/ST

In text languages IL and ST, FFBs can be called in formal and in informal form. Details can be found in the *Reference manual*.

Example of a formal function call:

```
out:=LIMIT (MN:=0, IN:=var1, MX:=5);
```

Example of an informal function call:

```
out:=LIMIT (0, var1, 5);
```

NOTE: The use of `EN` and `ENO` is only possible for formal calls.

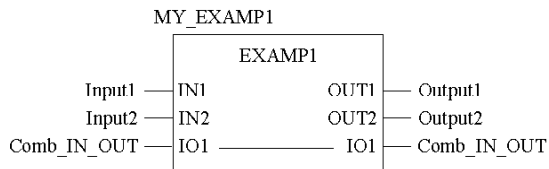
VAR_IN_OUT variable

FFBs are often used to read a variable at an input (input variables), to process it and to output the altered values of the **same** variable (output variables).

This special type of input/output variable is also called a `VAR_IN_OUT` variable.

The input and output variable are linked in the graphic languages (FBD and LD) using a line showing that they belong together.

Function block with `VAR_IN_OUT` variable in FBD:



Function block with `VAR_IN_OUT` variable in ST:

```
MY_EXAMP1 (IN1:=Input1, IN2:=Input2, IO1:=Comb_IN_OUT,
          OUT1=>Output1, OUT2=>Output2);
```

The following points must be considered when using FFBs with `VAR_IN_OUT` variables:

- The `VAR_IN_OUT` inputs must be assigned a variable.
- Literals or constants cannot be assigned to `VAR_IN_OUT` inputs/outputs.

The following additional limitations apply to the graphic languages (FBD and LD):

- When using graphic connections, `VAR_IN_OUT` outputs can only be connected with `VAR_IN_OUT` inputs.
- Only one graphical link can be connected to a `VAR_IN_OUT` input/output.
- Different variables/variable components can be connected to the `VAR_IN_OUT` input and the `VAR_IN_OUT` output. In this case the value of the variables/variable component on the input is copied to the output variables/variable component.
- No negations can be used on `VAR_IN_OUT` inputs/outputs.
- A combination of variable/address and graphic connections is not possible for `VAR_IN_OUT` outputs.

EN and ENO

Description

An **EN** input and an **ENO** output can be configured for the FFBs.

If the value of **EN** is equal to "0" when the FFB is invoked, the algorithms defined by the FFB are not executed and **ENO** is set to "0".

If the value of **EN** is equal to "1" when the FFB is invoked, the algorithms defined by the FFB will be executed. After the algorithms have been executed successfully, the value of **ENO** is set to "1". If certain error conditions are detected when executing these algorithms, **ENO** is set to "0".

If the **EN** pin is not assigned a value, when the FFB is invoked, the algorithm defined by the FFB is executed (same as if **EN** equals to "1"), Please refer to *Maintain output links on disabled EF* (see *Unity Pro, Operating Modes*).

If the algorithms are executed successfully, then value of **ENO** is set to "1", else **ENO** is set to "0".

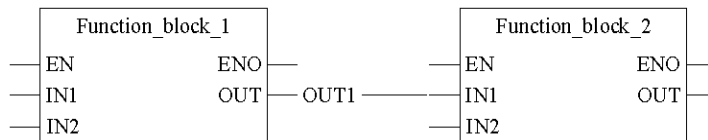
If **ENO** is set to "0" (caused by **EN**=0 or a detected error condition during execution or unsuccessful algorithm execution):

- Function blocks
 - EN/ENO handling with function blocks that (only) have one link as an output parameter:



If **EN** from **FunctionBlock_1** is set to "0", the output connection **OUT** from **FunctionBlock_1** retains the status it had in the last correctly executed cycle.

- EN/ENO handling with function blocks that have one variable and one link as output parameters:



If **EN** from **FunctionBlock_1** is set to "0", the output connection **OUT** from **FunctionBlock_1** retains the status it had in the last correctly executed cycle. The variable **OUT1** on the same pin, either retains its previous status or can be changed externally without influencing the connection. The variable and the link are saved independently of each other.

- Functions/Procedures

⚠ CAUTION

UNEXPECTED BEHAVIOR OF EQUIPMENT

For Unity Pro V4.0 and earlier versions, do not use links to connect function blocks outputs, when your application relies on persistent output data of an EF.

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

NOTE: With Unity Pro V4.0 and earlier versions the deactivation of an EF (EN=0) causes links connected to its Input/Output to be reset. To transfer the state of the signal do not use a link. A variable must be connected to the EF's output and must be used to connect the input of the element. With Unity Pro V4.1 and later versions you can maintain the output links even if an EF is deactivated by activating the option **Maintain output links on disabled EF (EN=0)** via the menu **Tools → Program → Languages → Common**.

As defined in IEC61131-3, the outputs from deactivated functions (EN-input set to "0") is undefined. (The same applies to procedures.)

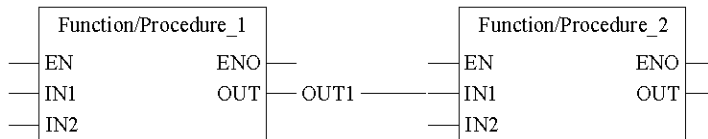
Here is an explanation of the output status in this case:

- EN/ENO handling with functions/procedures that (only) have one link as an output parameter:



If EN from Function/Procedure_1 is set to "0", the output connection OUT from Function/Procedure_1 is also set to "0".

- EN/ENO handling with function blocks that have one variable and one link as output parameters:



If EN from Function/Procedure_1 is set to "0", the output connection OUT from Function/Procedure_1 is also set to "0". The variable OUT1 on the same pin, either retains its previous status or can be changed externally without influencing the connection. The variable and the link are saved independently of each other.

The output behavior of the FFBs does not depend on whether the FFBs are called up without EN/ENO or with EN=1.

Conditional/Unconditional FFB Call

"Unconditional" or "conditional" calls are possible with each FFB. The condition is realized by pre-linking the input **EN**.

- **EN** connected
conditional calls (the FFB is only processed if **EN** = 1)
- **EN** shown, hidden, and marked TRUE, or shown and not occupied
unconditional calls (FFB is processed independent from **EN**)

NOTE: For disabled function blocks (**EN** = 0) with an internal time function (e.g. DELAY), time seems to keep running, since it is calculated with the help of a system clock and is therefore independent of the program cycle and the release of the block.

CAUTION

UNEXPECTED APPLICATION EQUIPMENT

Do not disable function blocks with internal time function during their operation.

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

Note for IL and ST

The use of **EN** and **ENO** is only possible in the text languages for a formal FFB call, e.g.

```
MY_BLOCK (EN:=enable, IN1:=var1, IN2:=var2,  
ENO=>error, OUT1=>result1, OUT2=>result2);
```

Assigning the variables to **ENO** must be done with the operator **=>**.

With an informal call, **EN** and **ENO** cannot be used.

Chapter 2

Block Availability on Various Hardware Platforms

Availability of the Blocks on the Various Hardware Platforms

Introduction

Not all blocks are available on all hardware platforms. The blocks which are available on your hardware platform can be found in the following tables.

Advanced

Availability of the blocks:

Block Name	Block Type	M340	M580	Momentum	Premium	Quantum
FCT_ACCEPT	Procedure	-	-	-	+	-
FCT_BIND	Procedure	-	-	-	+	-
FCT_BIND_UINT	Procedure	-	-	-	+	-
FCT_CLOSE	Procedure	-	-	-	+	-
FCT_CONNECT	Procedure	-	-	-	+	-
FCT_CONNECT_UINT	Procedure	-	-	-	+	-
FCT_LISTEN	Procedure	-	-	-	+	-
FCT_RECEIVE	Procedure	-	-	-	+	-
FCT_SELECT	Procedure	-	-	-	+	-
FCT_SEND	Procedure	-	-	-	+	-
FCT_SETSOCKOPT	Procedure	-	-	-	+	-
FCT_SHUTDOWN	Procedure	-	-	-	+	-
FCT_SOCKET	Procedure	-	-	-	+	-
Legend:						
+		available				
-		not available				

Part II

Introduction to TCP Open

Overview

This part introduces the TCP Open service which can be used with Elementary Functions (EF) in Unity Pro.

What Is in This Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
3	General	25
4	Warnings	27
5	Description of Operation	29
6	Operating Modes and Performance	45

Chapter 3

General

General Points and Principles of TCP Open

At a Glance

TCP Open for Premium is a set of Elementary Functions (EFs) and Derived Function Blocks (DFBs) that provide TCP/IP services in an automation application on Premium PLCs.

The TCP Open EFs and DFBs are installed from a CD. Once installed in Unity Pro, they appear in the **TCP Open** library in the **Advanced** family.

These preset functions can be used for TCP/IP Client/Server applications without having to have any knowledge of programming languages such as C ++ or Java.

Programming is carried out directly using the EFs and DFBs in the desired automation language (ST, LD, FBD or IL).

This TCP Open is available on the following modules:

- TSX ETY 110WS
- TSX ETY 5103

This documentation contains a description of the EFs for TCP Open; the documentation concerning the DFBs for TCP Open is supplied on the installation CD.

Chapter 4

Warnings

Notes and Warnings

At a Glance

Implementing TCP/IP services using TCP Open EFs is relatively straightforward. However, there are certain prerequisites for such an implementation, which is governed by the same development principles as a standard automation application.

Observations

To use the TCP Open EF library you must have a minimum working knowledge of TCP/IP applications and of how a socket system operates.

The implementation of Client/Server services follows certain programming rules which you must understand.

WARNING

UNINTENDED EQUIPEMENT OPERATION

The implementation and maintenance of TCP Open should only be performed by qualified personnel, with the necessary skills to handle sockets. This document should not be considered to represent adequate training for someone who is not otherwise qualified to develop TCP/IP services.

Although all reasonable precautions have been taken to ensure that the information provided here is accurate and authoritative, no responsibility is assumed by Schneider Electric for any consequences arising from the use of this document.

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

Using the TCP Open V2.0 Function Library

When using DFBs from version V1 with Unity V2.0, you may encounter the following malfunction: when the project is compared with the library the "Object code different" error message is displayed and you are asked to update your DFBs. Due to a checksum error, updating the DFBs will always generate this message when performing subsequent comparisons with the library. In spite of this, the DFBs still function correctly and can be used without restriction whether you are importing the program from Unity 1.0 or using DFBs from CD.

This malfunction will be corrected in a future version of TCP Open > V2.0.

Responsibilities

Schneider Electric is not liable for:

WARNING

UNEXPECTED SYSTEM BEHAVIOR

The design of a TCP Opensystem implies:

- To check and validate the design of the communication system architecture (client/server operating modes and protocols, performances, etc.)
- To check the implementation of appropriate EFs (or reuse of example EFs included in the TCP Open kit)
- To test and validate the EFs integrated into the communication system architecture
- To perform appropriate maintenance and manage diagnostics errors.

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

Chapter 5

Description of Operation

Subject of this Chapter

This chapter describes the principles governing the operations and implementation of a TCP/IP service using preset TCP Open EFs.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Operating Rules for TCP Open EFs	30
Principle Governing Socket Operation	31
General Structure of a TCP Open Communication EF	33
Structure of TCP Open Management Parameters	34
Management Parameters: Communication and Operation Reports	36
The Client/Server Model	38
Example of Client/Server Applications	40

Operating Rules for TCP Open EFs

Execution of a TCP Open EF

TCP Open EFs are executed asynchronously with the PLC cycle. Each EF call triggers a transaction with the Ethernet module concerned (TSX ETY 5103 or TSX ETY 110WS).

The transaction starts at the end of the PLC cycle and may take several cycles to complete. It is therefore necessary to manage the sequencing of calls so as not to saturate the module or request an action before the previous one is complete.

It is possible to call several TCP/IP services in the same PLC cycle. However, it is not certain that they will be processed in the chronological order in which they were called.

NOTE: We advise you to wait until the execution of one function is complete before requesting a new service on the same socket.

For example, wait for the `FCT_SOCKET` EF to be returned before calling the `FCT_BIND` EF, and wait for the `FCT_BIND` EF to be returned before calling the `FCT_LISTEN` EF.

EFs Supplied

The number of TCP Open functions supplied, as well as the manner in which they are used have been deliberately reduced in order to simplify the implementation of these services.

In addition to this, certain parameters are a requirement of the TSX ETY 5103 module. These limitations are detailed as part of the description of the `FCT_SOCKET` ([see page 91](#)) function.

Principle Governing Socket Operation

Introduction

The socket is the basic element of TCP communication. It is the socket which transports data.

The TCP/IP function library only provides sockets for flow management and connection between two devices.

NOTE: The TSX ETY 5103 Premium architecture can support up to 64 sockets. They can be used as listening (server) sockets or connected (client) sockets. In a server application, at least 1 socket must be a listening socket. There are no other hard boundaries with respect to the number of client connections versus server connections.

A socket can be created by either the `FCT_SOCKET` (see page 91) function or the `FCT_ACCEPT` (see page 53) function. The function returns a number that is used to access the socket.

Establishing a Server Connection

The following table describes the different steps that need to be created on the server in order to establish a connection.

Step	Action
1	Create a socket using the <code>FCT_SOCKET</code> function
2	Associate the created socket with an address (Port number and IP address) using the <code>FCT_BIND</code> (see page 57) function.
3	Set up the socket to accept connections using the <code>FCT_LISTEN</code> (see page 69) function. Note: the PLC acts as TCP server, the initial socket listens, and receives client socket connections.
4	Apply the <code>FCT_ACCEPT</code> (see page 53) function to this socket to create a new socket which will establish the connection. Note: this new socket is then connected to the client socket, and its number is returned by the <code>FCT_ACCEPT</code> function. The initial socket is then freed for other clients which wish to connect to the server.

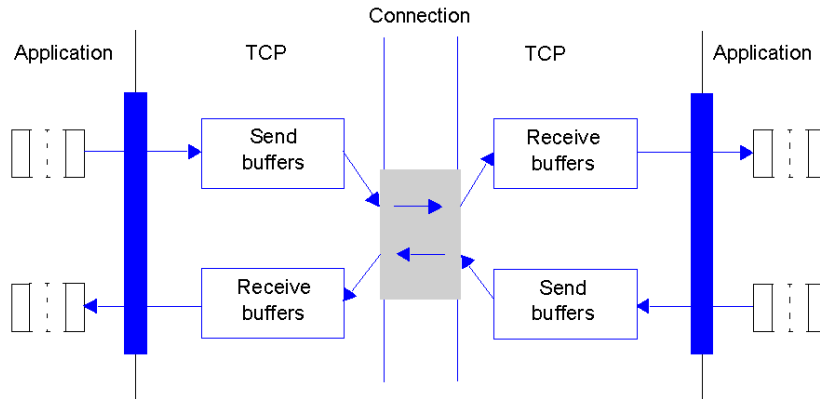
Establishing a Client Connection

The following table describes the different steps that need to be created on the client in order to establish a connection.

Step	Action
1	Create a socket using the <code>FCT_SOCKET</code> function block. The block returns a socket number which can then be used in the subsequent function blocks.
2	Use <code>FCT_CONNECT</code> to make a connection to another Ethernet device by specifying its IP address and the particular port that you are going to communicate on.

Exchanging Data over a TCP Connection

Once a connection has been established, data can be transferred. Transfers are carried out using the `FCT_SEND` (see page 79) and `FCT_RECEIVE` (see page 71) functions. The diagram below shows how these exchanges work.



Managing Sockets

Other functions can be used to act on sockets:

- `FCT_SETSOCKOPT` (see page 83): associates options with a socket. These options modify the behavior of the socket.
- `FCT_SELECT` (see page 75): used to test events on sockets.
- `FCT_SHUTDOWN` (see page 87): disables transmission on the socket.
- `FCT_CLOSE` (see page 61): frees the socket descriptor when it is no longer used.

General Structure of a TCP Open Communication EF

At a Glance

Communication functions are processed asynchronously in relation to the PLC cycle. A function is executed over several consecutive cycles following the cycle which launched its execution.

The parameters of a TCP Open communication function are the following:

- interface number
- specific parameters
- management parameters

Interface Number

This interface number corresponds to the slot number of the ETY module in the main rack.

NOTE: Only rack 0 can take an Ethernet module that uses TCP Open communication functions.

In the case of the TSXETY5103, the Premium architecture divides this integer into 2 bytes.

- The low byte contains the slot number of the sockets.
- The value in the high byte can be used to extend the number of sockets as follows:
 - 00 is used for full backward compatibility with applications created on firmware version 3.3 or earlier.
 - 01 indicates that up to 64 sockets can be used; the firmware must be higher than version 3.3 to support this socket extension.

NOTE: When you set the value of the high byte to 01 (to extend the number of sockets to 64), make sure that the TSXETY5103 firmware is at an acceptable version. If you attempt to use this high-byte setting when the firmware is at version 3.3 or lower, the module goes into a constant reboot cycle and does not become operational.

Specific Parameters

These parameters are specific to each function. There may be more than one, in which case they are separated by commas. They are described in the chapters specific to each function.

Management Parameters

The management parameters consist of an array of 4 integers also known as the Management table ([see page 34](#)). It is identical to the management table for standard communication EFs except for the exchange number which is not managed and the Time out.

Structure of TCP Open Management Parameters

At a Glance

The management parameters are grouped together in a array of four integers. The values contained in this array are used to control the communication functions.

NOTE: In the technical documentation, these parameters are also known as a management table or report.

Structure

The table below describes the data structure of the communication management table:

Word Order	Most Significant Byte	Least Significant Byte
1	Reserved	Activity bit
2	operation report	communication report
3	Reserved	
4	Length	

Activity Bit

This bit signals the execution status of the communication function.

It is set to 1 when launched and falls back to 0 when execution is complete.

This is the first bit of the first element in the table.

Example: If the management table has been declared in the following way:

`Tab_Gest ARRAY [1..4] OF INT`, the activity bit is the bit with the notation `Tab_Gest[1].0`.

NOTE: The notation used above requires Unity Pro to be used in non-IEC mode. If this is not the case `Tab_Gest[1].0` cannot be accessed in this manner.

Operation Report

Operation reports are described in *Operation Report*, [page 37](#).

Communication Report

Communication reports are described in *Communication Report*, [page 36](#).

Length

The length parameter is used with the `FCT_SEND` ([see page 79](#)) and `FCT_RECEIVE` ([see page 71](#)) functions. The length parameter is also used with the `FCT_SELECT` ([see page 65](#)) function. With `FCT_SELECT` the length field is not used when the high byte of the `INTE` parameter is set to 00. You should set the length to 8 when the high byte of the `INTE` parameter is set to 01. A setting of 8 allows you to see the data associated with all 64 connections. If you set the value lower, say to 4, you will see only the data associated with the first 32 connections.

Management Parameters: Communication and Operation Reports

At a Glance

Communication and operation reports are part of the management parameters.

NOTE: It is recommended that you always test communication function reports as soon as their execution is complete and before they are re-activated. On a cold restart, you must under all circumstances check that all the management parameters of the communication functions have been reset to 0.

Communication Report

This report is common to all functions. It is significant when the activity bit changes from 1 to 0.

Reports whose value is between 16#01 and 16#FE concern errors detected by the processor which executed the function.

The different values of this report are indicated in the following table:

Value	Communication Report (Least Significant Byte)
16#00	Exchange successful
16#01	Exchange stopped on timeout NOTE: : When a Modbus M340 CPU sends a MODBUS BROADCAST request, we obtain this error code despite the broadcast is well done.
16#05	Incorrect management parameter format
16#06	Incorrect specific parameters
16#07	Network module missing or incorrect address
16#0B	No system resources: the number of simultaneous communication EFs exceeds the maximum that can be managed by the processor
16#0E	Incorrect send length
16#FF	Message refused NOTE: OK value returned when a TSX SCP ... / TSX SCY ... sends a MODBUS BROADCAST request

NOTE: The function can detect an error in the parameters before activating the exchange. In this case, the activity bit remains at 0 and the report is initialized with the values corresponding to the fault.

Operation Report

This report byte describes the result of interaction of the function with the TCP/IP stack of the network module.

It is significant only if the communication report has the following values:

- 16#00 (exchange successful)
- 16#FF (message refused).

If this report has a value equal to 16#00, the operation report is specific to each function. It is described in the chapters devoted to these functions.

If the communication report has the value 16#FF, the operation report has the following values:

Value	Operation Report (Most Significant Byte)
16#0B	No system resources: the number of simultaneous communication EFs exceeds the maximum that can be managed by the processor
16#0C	Network module not running

The Client/Server Model

At a Glance

The client/server model consists of two entities, one of which acts as a **server**, and which responds to requests and the other of which is a **client**, which makes the request.

The client/server model operates according to a specific protocol which must be implemented at either end of the connection.

This protocol is described in the following paragraphs.

NOTE: If you are developing your own client/server application, you are responsible for testing and managing the connections.

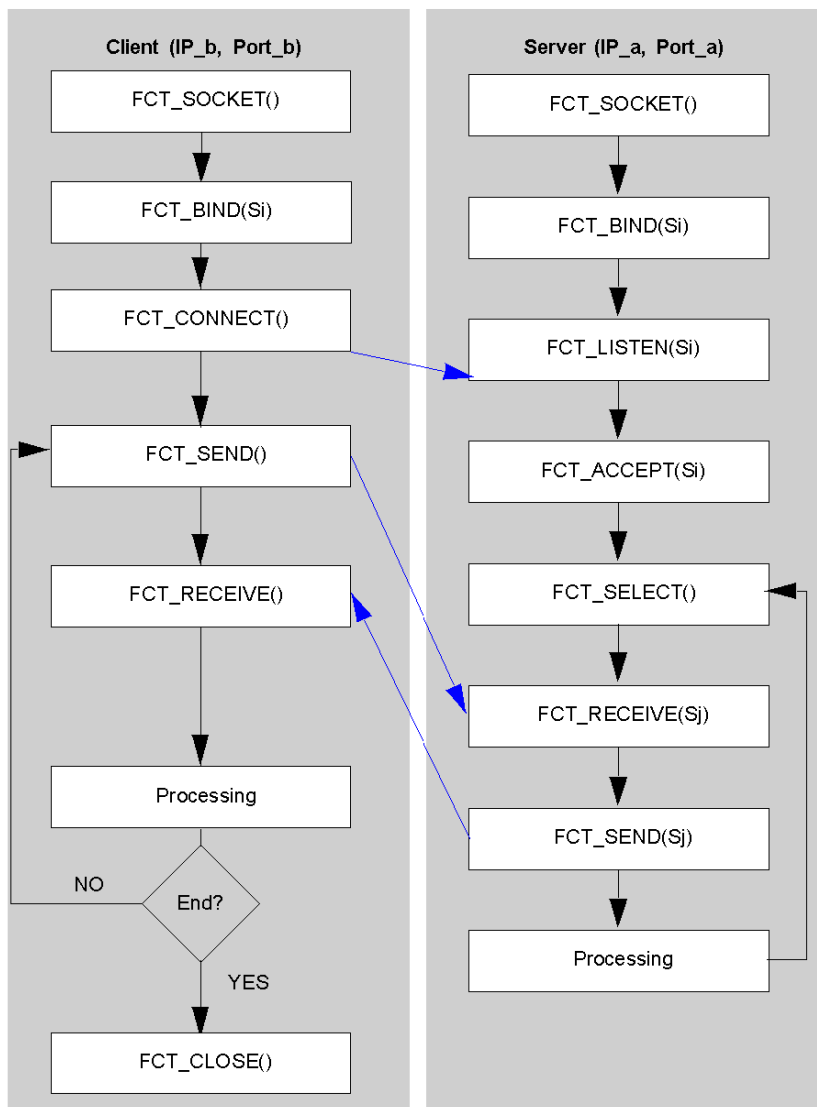
Description

The model works in the following way:

- The server application listens.
- A client application requests services from the server application.
- The server application accepts the request.
- Exchanges are made between the two entities.

Illustration

Illustration of how the client/server model works



Example of Client/Server Applications

At a Glance

Connections are managed by the application program. A client which terminates a connection without observing the correct TCP sequence (as a result of a power outage, for example), is considered as still being connected as long as nothing else has been sent or any other events have occurred.

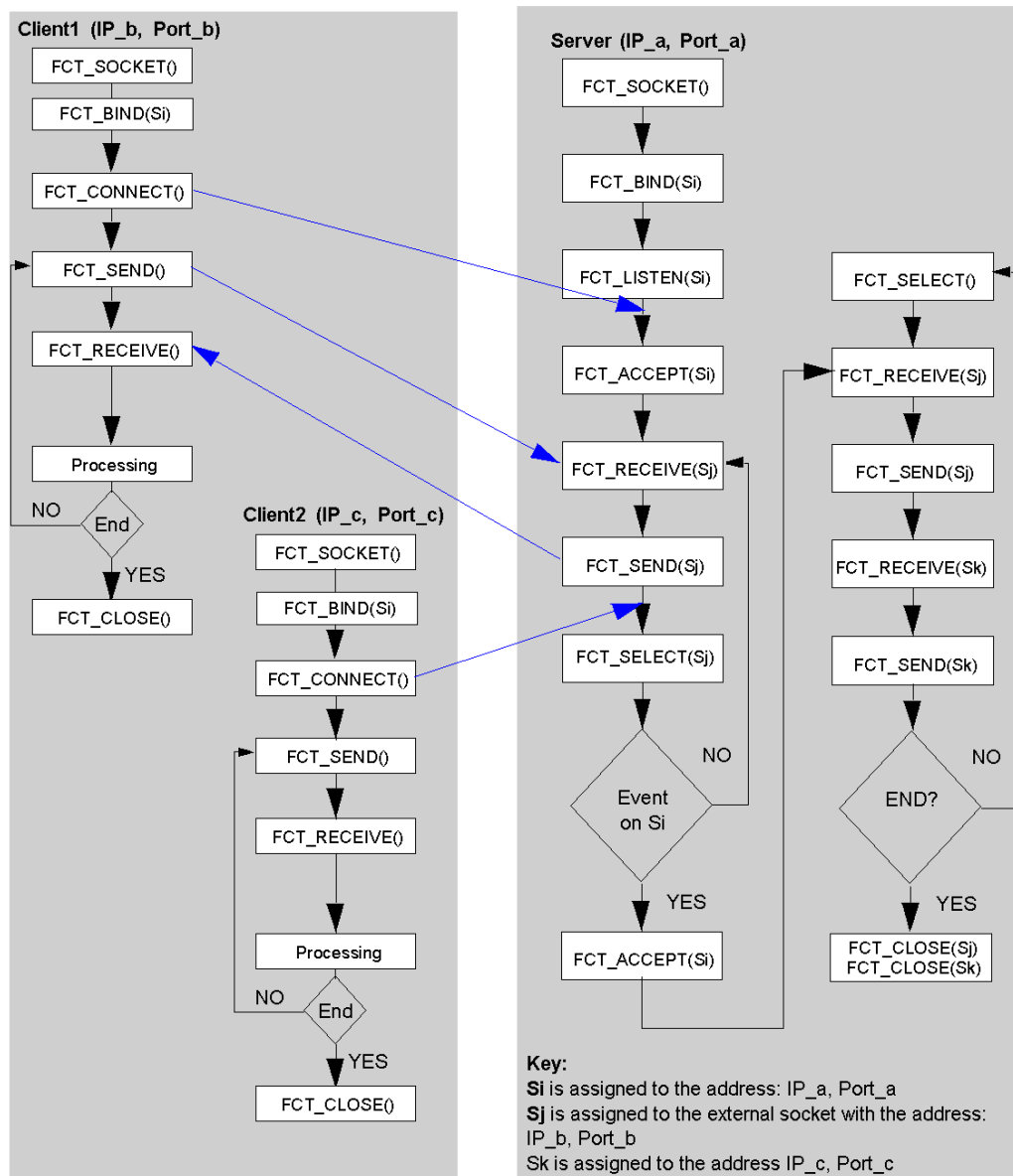
The following paragraphs provide three significant examples which should enable you to better understand how the architecture works.

The client side is described in the form of a flow diagram, whereas on server side is shown as a sequence of operations closely linked to the events on the client side.

The scenario in example 2 is of a connection cut-off on the client side after `FCT_RECEIVE (Sj)`, as there is no `SEND_RECEIVE`-processing loop.

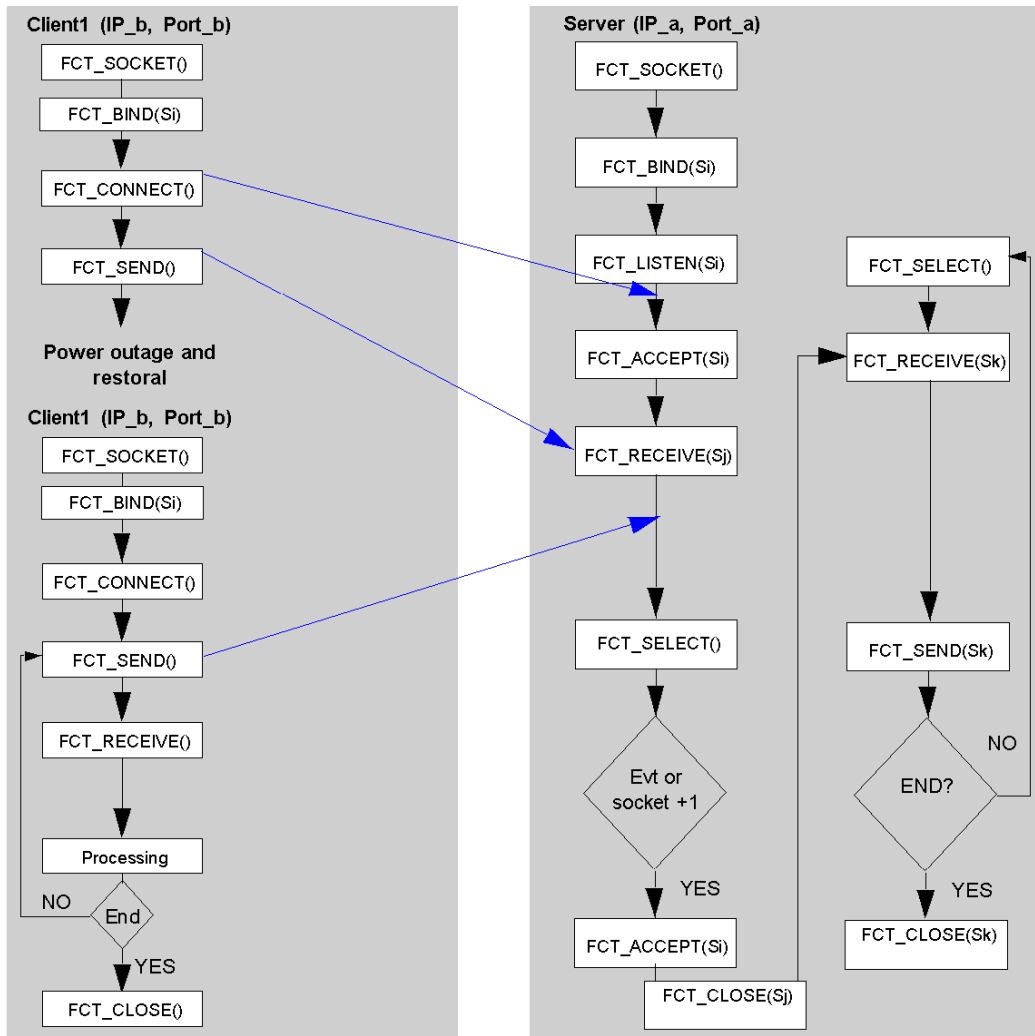
Example 1

The diagram below describes how a server application operates when processing two connections requested by two clients.



Example 2

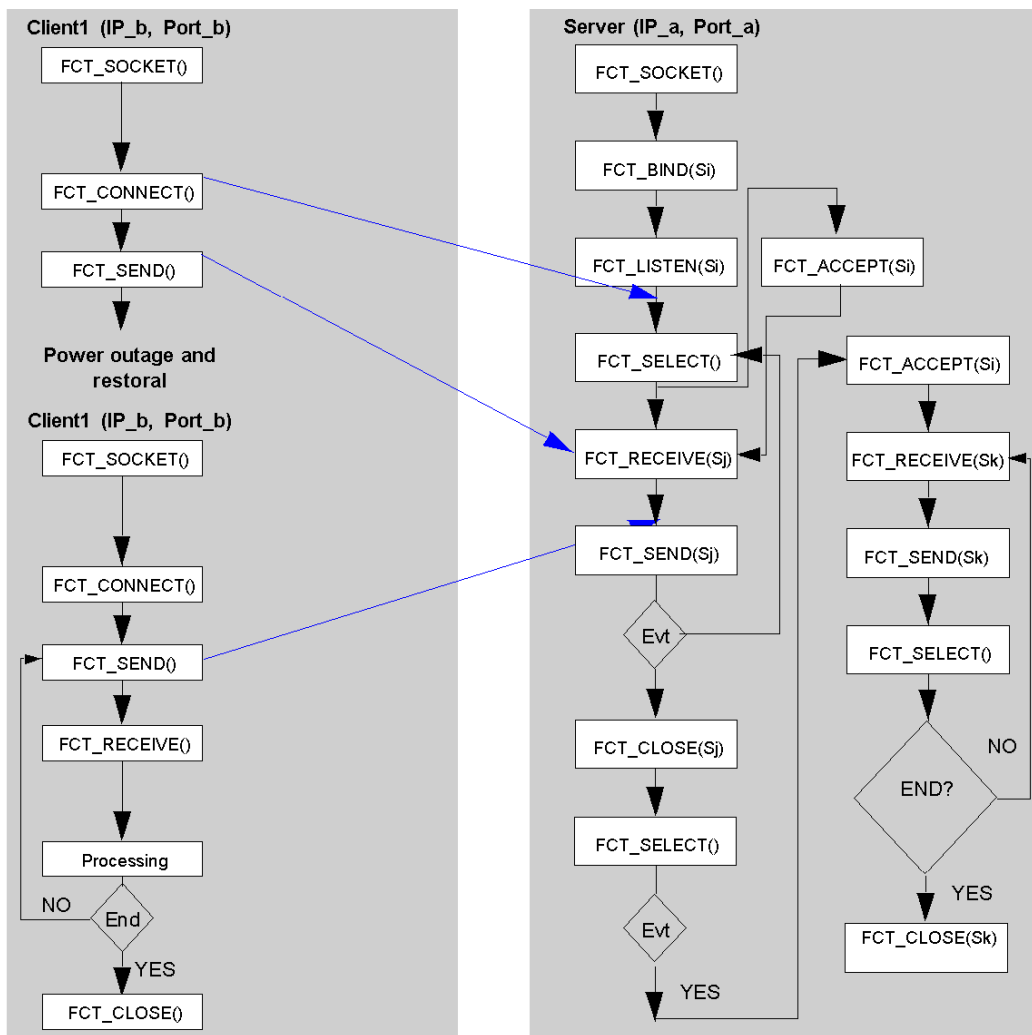
The diagram below describes how a server application operates when processing two connections requested by the same client. The first disconnection was not made in accordance with correct TCP procedure (e.g.: power failure). The socket is considered to still be connected as long as the client has not made a new connection.



Key:
Si is assigned to the address: IP_a, Port_a
Sj is assigned to the external socket with the address: IP_b, Port_b
Sk is assigned to the address IP_b, Port_b, a new socket is created on receiving the second request from the client.

Example 3

The diagram below describes how a server application operates when processing two connections requested by the same client. The first disconnection was not made in accordance with correct TCP procedure (e.g.: power failure). As long as nothing has been sent by the server, the socket is still considered to be connected.



Chapter 6

Operating Modes and Performance

Subject of this Chapter

This chapter is intended to provide you with an introduction to the operating modes, the basic notions required for debugging and the performance characteristics of the TSX ETY 5103 module (Ethernet module supporting TCP open functions).

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Operating Modes of the Network Module	46
Performance	48
Debugging and Diagnostics	49

Operating Modes of the Network Module

At a Glance

The TSX ETY 5103 module has 4 operating states:

- power off
- self-test running
- configured
- not configured

Self-tests are performed on power-up.

The module does not operate with its default configuration.

The configuration must be sent to the PLC via the terminal socket and not over the network.

If the module has not been correctly configured, it cannot process the sockets and a refuse message is sent back to the TCP Open functions.

Sending the Configuration to the Module

The configuration is sent to the module in the following cases:

- when an application is downloaded
- when the PLC is powered up
- when the module is connected to the rack when the power is on
- on a warm or cold restart
- when the Premium is reset

Operating Mode after Configuration

WARNING

UNINTENDED APPLICATION BEHAVIOR - WARM OR COLD RESTART

The programmer must test the system bits %S0 and %S1 in his application in order to re-create the connections if a warm or cold start has been performed. The programmer must also test the system bit %S13 in his application in order to check for first scan after a STOP/RUN command using the PLC software application.

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

Once a configuration has been defined, the module operates in the following way:

Step	Action
1	The module resets the communication in progress. Result: Exchanges in progress are ended; TCP Open connections are closed, and all sockets are deleted.
2	The module reconfigures itself.
3	The module is ready to process the TCP Open communication functions of the application.

Performance

Number of Simultaneous Connections

The maximum number of simultaneous TCP/IP connections to a TSX ETY 5103 is:

- 32 if the high byte of the INTE is set to 00; 16 can be connected (client) sockets and 16 can be listening (server) sockets.
- 64 if the high byte of the INTE parameter is set to 01. The 64 socket can be used as listening (server) sockets or connected (client) sockets in any combination with one exception—in a server application, at least 1 socket must be a listening socket.

Data Exchanges

The maximum chunk of data that can be sent in one PLC cycle is 240 bytes. This limitation is due to the X-bus mechanism for data transfer between the module and the processor.

If you wish to transfer a message of 8 Kbytes, you must break down your message into blocks of 240 bytes. If you want to guarantee the order in which the blocks are sent so as to be able to reconstitute the whole message, you must send one block for each cycle. That means that 35 PLC cycles ($8 \times 1024 / 240$) would be required. A PLC cycle of 50 ms would take 1.75 s.

NOTE: These calculations are based on the use of a single socket. If you are managing several clients, you must take into account the number of connected sockets.

For a message oriented protocol, a lower level interface has to manage the fragmentation process. Here performance depends on the number of FCT_SEND ([see page 79](#)) or FCT_RECEIVE ([see page 71](#)) functions executed in the same PLC cycle.

Performance can be reduced depending on the degree to which the TSX ETY 5103 module is used for other communications tasks (IO Scanning, Global Data, etc.).

Debugging and Diagnostics

Debug Screen

In online mode, Unity Pro software can be used to debug the application using the application-specific debug screens.

The module debug screen can be used to do this. However, you should note that:

- The number of connections displayed includes:
 - open profile connections (TCP Open)
 - private profile connections (address declared in configuration)
- The IP addresses of open profiles cannot be seen in this screen.

IP Communication Tests

You can use the communication test window to test IP communication with client devices if the IP address of the client is declared as a remote device (used by the private profile).

The list of IP addresses configured is used to select the station with which to communicate by activating a "ping", which feeds back as a status the loop-back or the time out of the message.

Part III

Advanced

Overview

This part describes the elementary functions and elementary function blocks of the `Advanced` family.

What Is in This Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
7	FCT_ACCEPT: Accepts a Connection Request	53
8	FCT_BIND: Binds a Socket Number to an IP Address and a Port	57
9	FCT_CLOSE: Deletes the Specified Socket	61
10	FCT_CONNECT: Establishes a Connection with an IP Address	65
11	FCT_LISTEN: Configuration of a Socket Await Connection	69
12	FCT_RECEIVE: Retrieves Data Available on a Socket	71
13	FCT_SELECT: Multiplexes Requests Over Sockets	75
14	FCT_SEND: Sending Data to a Specified Socket	79
15	FCT_SETSOCKOPT: Sets the Options Associated with the Socket	83
16	FCT_SHUTDOWN: Disables Transmission on the Socket	87
17	FCT_SOCKET: Creation of a New Socket	91

Chapter 7

FCT_ACCEPT: Accepts a Connection Request

Description

Function Description

The `FCT_ACCEPT` function is used to accept a connection request received by the specified socket. This connection request comes from a foreign socket.

Before `FCT_ACCEPT` is called, a socket must be set up to receive a connection request by issuing the `FCT_LISTEN` call. The `FCT_ACCEPT` function:

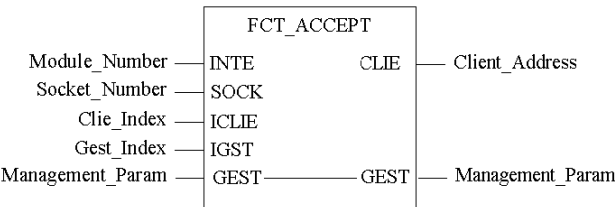
- extracts the first connection request in the queue of pending connections
- creates a connected socket with the same properties as the original socket
- completes the connection between the foreign socket and the new socket
- and returns a number for the new socket

The new returned socket number is used to read from and write data to the foreign socket. It is not used to accept more connections. The original socket remains open for accepting further connections.

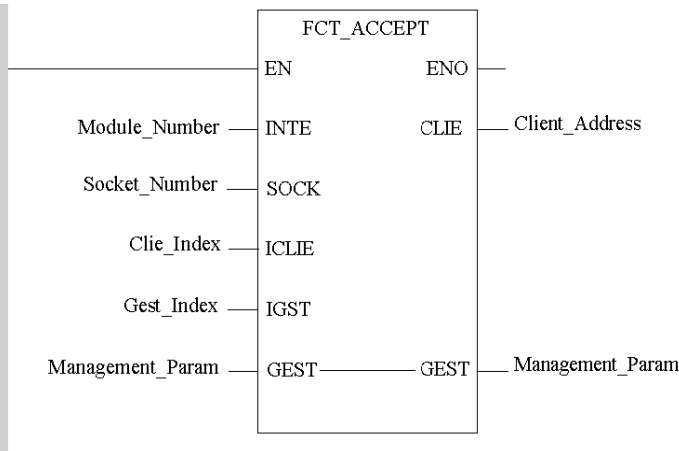
If there are no pending connections in the queue, `FCT_ACCEPT` returns an error.

The additional parameters `EN` and `ENO` may be configured.

Representation in FBD



Representation in LD



Representation in IL

```
LD Module_Number
FCT_ACCEPT Socket_Number, Clie_Index, Gest_Index, Management_Param, Client_Address
```

Representation in ST

```
FCT_ACCEPT(Module_Number, Socket_Number, Clie_Index, Gest_Index, Management_Param, Client_Address);
```

Description of Parameters

The following table describes the input parameters:

Parameter	Type	Comment
Module_Number	INT	Slot number of network module in rack 0. ● Low byte is the slot number of network module in rack 0. ● High byte can be used to extend the number of sockets. ● 00: Provided for full backward compatibility with applications created on firmware version 3.3 or earlier. ● 01: Up to 64 sockets can be used (firmware version must be higher than 3.3).
Socket_Number	INT	Socket number
Clie_Index	INT	Index of first word in Client_Address array
Gest_Index	INT	Index of first word in Management_Param array

The following table describes the output parameters:

Parameter	Type	Comment
Client_Address	ARRAY [0... 3] OF INT	Array of 4 words containing the service socket number, the port number and the IP address of the client: • Client_Address[0]: connected socket number • Client_Address[1]: client port number • Client_Address[2]: least significant word of the client IP address • Client_Address[3]: most significant word of the client IP address

The following table describes the input/output parameters:

Parameter	Type	Comment
Management_Param	ARRAY [0... 3] OF INT	Function management array (see page 34) The operation report can have the following values: • 16#00: no error • 16#09: the socket number is invalid • 16#16: the FCT_LISTEN function must be called before FCT_ACCEPT • 16#23: no connection request

Chapter 8

FCT_BIND: Binds a Socket Number to an IP Address and a Port

Description

Function Description

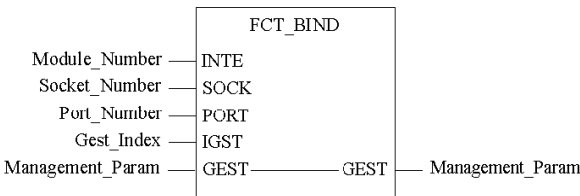
The `FCT_BIND` function is used to assign a port number and an internet address to a socket.

A socket is created without an address and cannot be used to receive data (except for connection requests) until it is assigned one. The internet address is fixed by the network module to its local configured IP address. The user is not allowed to use some port numbers because they are already used by the network module. These port numbers are:

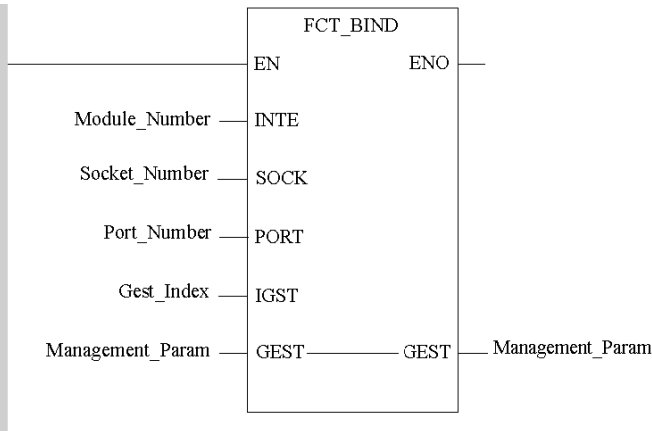
- 20 and 21 (FTP ports)
- 23 (Telnet port)
- 67 and 68 (BOOTP DHCP ports)
- 80 (HTTP port)
- 161 and 162 (SNMP ports)
- 502 (Schneider Electric port)
- 5000 and 5001 (specific ports of module)
- 1024 (TCP USER ports)
- 3124 (I/O port)
- 7400-8400 (RTPS ports)

The additional parameters `EN` and `ENO` may be configured.

Representation in FBD



Representation in LD



Representation in IL

```
LD Module_Number
FCT_BIND Socket_Number, Port_Number, Gest_Index, Management_Param
```

Representation in ST

```
FCT_BIND(Module_Number, Socket_Number, Port_Number, Gest_Index, Management_Param);
```

Description of Parameters

The following table describes the input parameters:

Parameter	Type	Comment
Module_Number	INT	Slot number of network module in rack 0. - Low byte is the slot number of network module in rack 0. - High byte can be used to extend the number of sockets. 00: Provided for full backward compatibility with applications created on firmware version 3.3 or earlier. 01: Up to 64 sockets can be used (firmware version must be higher than 3.3).
Socket_Number	INT	Socket number
Port_Number	INT	Port number to be assigned to socket
Gest_Index	INT	Index of first word in Management_Param array

The following table describes the input/output parameters:

Parameter	Type	Comment
Management_Param	ARRAY [0... 3] OF INT	Function management array (see page 34) The operation report can have the following values: ● 16#00: No error. ● 16#09: The socket number is invalid. ● 16#16: The socket is already bound. ● 16#30: The specified port is already in use. ● 16#37: The specified port number is not available. ● 16#41: No route to host.

Configuring Port Numbers Above 32K

The data type of the port parameter is an integer. In order to use port numbers higher than 15 bits (higher than 32K), you must use the `UINT_TO_INT` function block to convert the `Unsigned Integer` format (encoded in 16 bits) to an `Integer` format. Use the `UINT_TO_INT` function block before using `EF FCT_BIND` or `FCT_CONNECT` when a port number higher than 32K is used.

Input parameter:

Parameter	Data Type	Meaning
UINT_variable	UINT	input value

Output parameter:

Parameter	Data Type	Meaning
ConvertedVariable	INT	output value

Chapter 9

FCT_CLOSE: Deletes the Specified Socket

Description

Function Description

The `FCT_CLOSE` function deletes the specified socket.

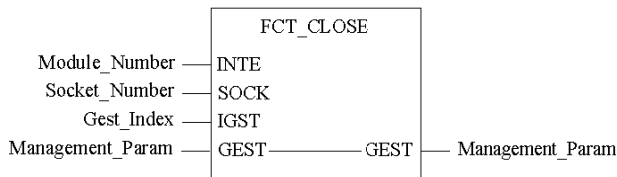
NOTE: If the socket number is not indicated or is 0, all open sockets are deleted.

As the sockets were opened with the `SO_LINGER` option when using `FCT_SOCKET`, the `FCT_CLOSE` function is not blocked, even if the queues have not yet been sent or been acknowledged. This is called a hard or abortive close, because the socket's virtual circuit is reset immediately, and any unsent data is lost.

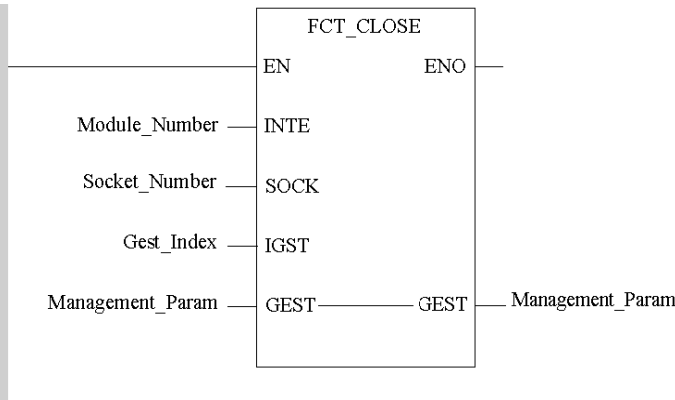
Any call to the `FCT_RECEIVE` function to the other side of the circuit will fail with the error message: `connection reset (16#36)`.

The additional parameters `EN` and `ENO` may be configured.

Representation in FBD



Representation in LD



Representation in IL

```
LD Module_Number
FCT_CLOSE Socket_Number, Gest_Index, Management_Param
```

Representation in ST

```
FCT_CLOSE(Module_Number, Socket_Number, Gest_Index, Management_Param);
```

Description of Parameters

The following table describes the input parameters:

Parameter	Type	Comment
Module_Number	INT	Slot number of network module in rack 0. • Low byte is the slot number of network module in rack 0. • High byte can be used to extend the number of sockets. • 00: Provided for full backward compatibility with applications created on firmware version 3.3 or earlier. • 01: Up to 64 sockets can be used (firmware version must be higher than 3.3).
Socket_Number	INT	Number of socket to be deleted. If the value of the <code>Socket_Number</code> is 0, all sockets are deleted.
Gest_Index	INT	Index of first word in <code>Management_Param</code> array

The following table describes the input/output parameters:

Parameter	Type	Comment
Management_Param	ARRAY [0... 3] OF INT	Function management array (see page 34) The operation report can have the following values: • 16#00: No error. • 16#16: The socket number is invalid.

Chapter 10

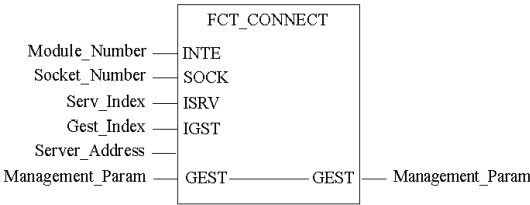
FCT_CONNECT: Establishes a Connection with an IP Address

Description

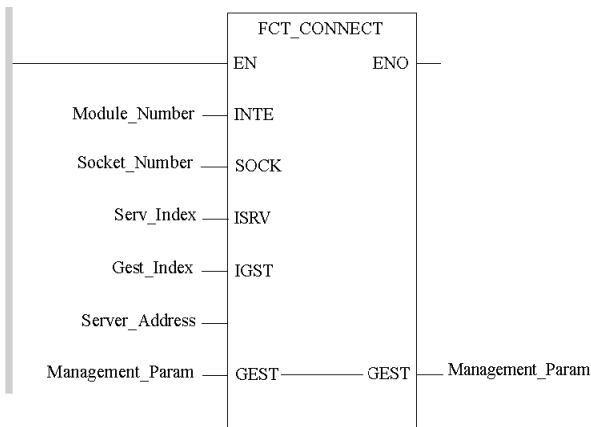
Function Description

The `FCT_CONNECT` function is used to establish a connection to a known port and internet address. The additional parameters `EN` and `ENO` may be configured.

Representation in FBD



Representation in LD



Representation in IL

```
LD Module_Number
```

```
FCT_CONNECT Socket_Number, Serv_Index, Gest_Index, Server_Address, Mana  
gement_Param
```

Representation in ST

```
FCT_CONNECT(Module_Number, Socket_Number, Serv_Index, Gest_Index, Serve  
r_Address, Management_Param);
```

Description of Parameters

The following table describes the input parameters:

Parameter	Type	Comment
Module_Number	INT	Slot number of network module in rack 0. • Low byte is the slot number of network module in rack 0. • High byte can be used to extend the number of sockets. • 00: Provided for full backward compatibility with applications created on firmware version 3.3 or earlier. • 01: Up to 64 sockets can be used (firmware version must be higher than 3.3).
Socket_Number	INT	Socket number
Serv_Index	INT	Index of first word in <i>Server_Address</i> array
Gest_Index	INT	Index of first word in <i>Management_Param</i> array
Server_Address	INT	Array of 3 words containing the port number and IP address of the server

The following table describes the input/output parameters:

Parameter	Type	Comment
Management_Param	ARRAY [0... 3] OF INT	Function management array (see page 34) The operation report can have the following values: • 16#00: Correct operation. • 16#09: Invalid socket number. • 16#16: Invalid parameter. • 16#20: Connection is cut. • 16#24: Connection in progress. • 16#38: Socket already connected. • 16#3D: Connection refused. • 16#41: No route to host.

Configuring Port Numbers Above 32K

The data type of the port parameter is an integer. In order to use port numbers higher than 15 bits (higher than 32K), you must use the `UINT_TO_INT` function block to convert the `Unsigned Integer` format (encoded in 16 bits) to an `Integer` format. Use the `UINT_TO_INT` function block before using `EF_FCT_BIND` or `FCT_CONNECT` when a port number higher than 32K is used.

Input parameter:

Parameter	Data Type	Meaning
UINT_variable	UINT	input value

Output parameter:

Parameter	Data Type	Meaning
ConvertedVariable	INT	output value

Chapter 11

FCT_LISTEN: Configuration of a Socket Await Connection

Description

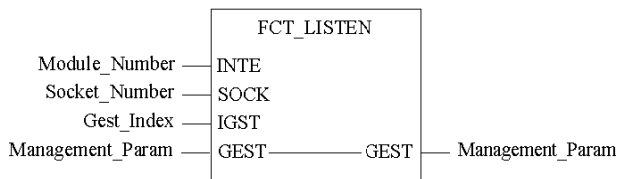
Function Description

The `FCT_LISTEN` function sets up the specified socket to receive connections.

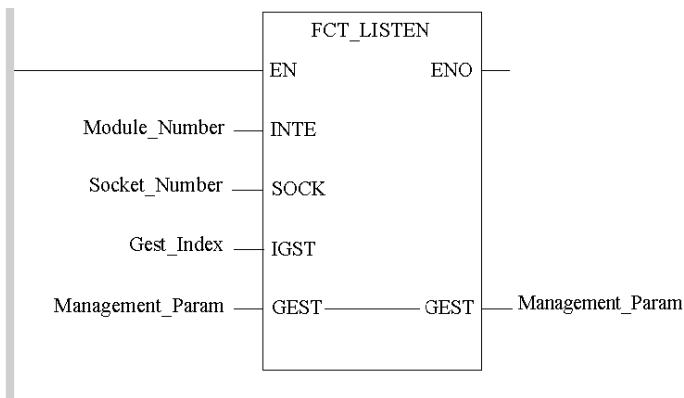
Connection requests are queued on the socket until they are accepted with the `FCT_ACCEPT` call. The length of the queue is set to 16. If a connection request arrives while the queue is full, the requesting client gets an `ECONNREFUSED (16#3D)` error.

The additional parameters `EN` and `ENO` may be configured.

Representation in FBD



Representation in LD



Representation in IL

```
LD Module_Number
```

```
FCT_LISTEN Socket_Number, Gest_Index, Management_Param
```

Representation in ST

```
FCT_LISTEN(Module_Number, Socket_Number, Gest_Index, Management_Param);
```

Description of Parameters

The following table describes the input parameters:

Parameter	Type	Comment
Module_Number	INT	Slot number of network module in rack 0. ● Low byte is the slot number of the network module in rack 0. ● High byte can be used to extend the number of sockets. ● 00: Provided for full backward compatibility with applications created on firmware version 3.3 or earlier. ● 01: Up to 64 sockets can be used (firmware version must be higher than 3.3).
Socket_Number	INT	Socket number
Gest_Index	INT	Index of first word in Management_Param array

The following table describes the input/output parameters:

Parameter	Type	Comment
Management_Param	ARRAY [0... 3] OF INT	Function management array (see page 34) The operation report can have the following values: ● 16#00: No error. ● 16#09: The socket number is invalid.

Chapter 12

FCT_RECEIVE: Retrieves Data Available on a Socket

Description

Function Description

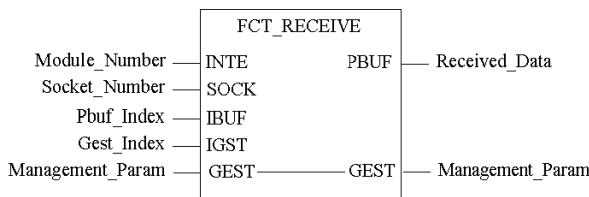
The `FCT_RECEIVE` function looks for the data available on the socket. The maximum length of data to read is 240 bytes.

It returns the numbers of bytes read in the socket. You should always test this value because it is the only way to check the actual number of data bytes stored in the user buffer.

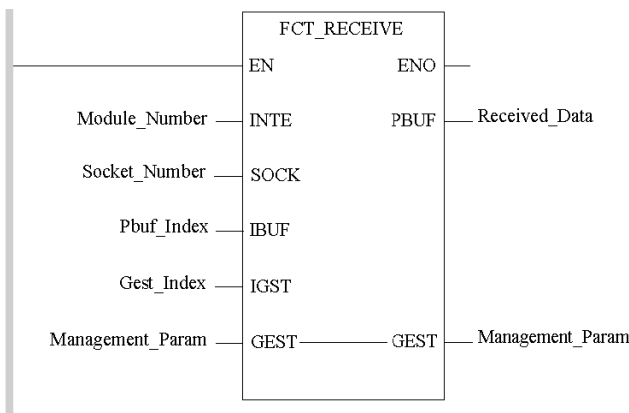
NOTE: `FCT_RECEIVE` does not return out-of-band data.

The additional parameters `EN` and `ENO` may be configured.

Representation in FBD



Representation in LD



Representation in IL

```
LD Module_Number
```

```
FCT_RECEIVE Socket_Number, Pbuf_Index, Gest_Index, Management_Param, Re  
ceived_Data
```

Representation in ST

```
FCT_RECEIVE(Module_Number, Socket_Number, Pbuf_Index, Gest_Index, Manag  
ement_Param, Received_Data);
```

Description of Parameters

The following table describes the input parameters:

Parameter	Type	Comment
Module_Number	INT	Slot number of network module in rack 0. ● Low byte is the slot number of the network module in rack 0. ● High byte can be used to extend the number of sockets. ● 00: Provided for full backward compatibility with applications created on firmware version 3.3 or earlier. ● 01: Up to 64 sockets can be used (firmware version must be higher than 3.3).
Socket_Number	INT	Socket number
Pbuf_Index	INT	Index of first word in Received_Data array
Gest_Index	INT	Index of first word in Management_Param array

The following table describes the output parameters:

Parameter	Type	Comment
Received_Data	ARRAY [0... n] OF INT	Array of maximum of 240 bytes containing the data read on the socket

The following table describes the input/output parameters:

Parameter	Type	Comment
Management_Param	ARRAY [0... 3] OF INT	Function management array (see page 34) The operation report can have the following values: ● 16#00: No error. ● 16#09: The socket number is invalid. ● 16#23: No data to read. ● 16#36: The connection has been reset by peer. ● 16#39: The socket is not connected (listening socket). ● 16#3C: The keep alive timed out on broken connection. ● 16#0E: The length of the character string to be received is greater than 240 bytes. The fourth word of the array should contain the number of bytes received if no error has occurred.

Chapter 13

FCT_SELECT: Multiplexes Requests Over Sockets

Description

Function Description

The `FCT_SELECT` function is used to multiplex I/O requests among multiple sockets. It indicates which sockets have events to process using an array of integers.

For ETY 5103 firmware revision 3.3 or lower:

- The socket descriptors are assigned a socket number from 1 to 32:
 - Numbers from 1 to 16 are assigned to sockets created by the `Socket` function. They are client sockets.
 - Numbers from 17 to 32 are assigned to sockets created by the `Accept` function. They are server sockets.
- The array of the `MASK (Socket_Activity)` is for diagnosing the status of the listening (server) and connected (client) sockets.
 - **Server Perspective:** Any word of the array uses available bits of the array to correspond to the listening socket in the initial step. The `Accept` function outputs a different socket number to locate its associated bit in the second word of the array, which reports the status of the server connection.
 - **Client Perspective:** The first word of the array reports the status of the client connection.

For ETY 5103 firmware greater than revision 3.3:

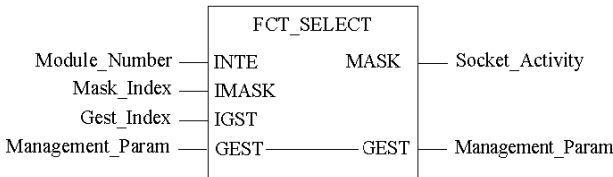
- You may have up to 64 sockets that can be used as listening (server) sockets or connected (client) sockets. In a server application, at least 1 socket must be a listening socket. There are no other hard boundaries regarding listening versus connected sockets.
- To determine the listening sockets, see the socket number provided as output by the `Socket` function.
- To determine the connected sockets, see the socket number provided as output by the `Accept` function.
- The array of the `MASK (Socket_Activity)` is for diagnosing the status of the listening and connected sockets, which are now combined into a 2- or 4-word array (see note below). You must use the output of the `Socket` or `Accept` function to locate the corresponding bit in the array. See the `Socket_Activity` in the Description of Parameters table ([see page 77](#)).

NOTE: For versions higher than 3.3, a length parameter is required in the `Management_Param` structure.

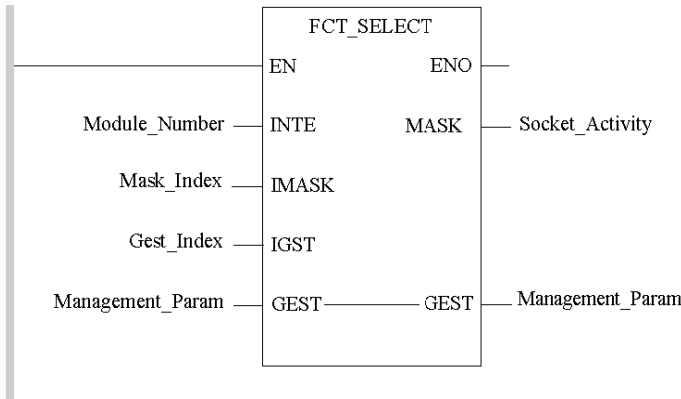
The length field is not used when the high byte of the INTE parameter is set to 00. You should set the length to 8 when the high byte of the INTE parameter is set to 01. A setting of 8 allows you to see the data associated with all 64 connections. If you set the value lower, say to 4, you will see only the data associated with the first 32 connections.

The additional parameters `EN` and `ENO` may be configured.

Representation in FBD



Representation in LD



Representation in IL

```
LD Module_Number
FCT_SELECT Mask_Index, Gest_Index, Management_Param, Socket_Activity
```

Representation in ST

```
FCT_SELECT(Module_Number, Mask_Index, Gest_Index, Management_Param, Socket_Activity);
```

Description of Parameters

The following table describes the input parameters:

Parameter	Type	Comment
Module_Number	INT	Slot number of network module in rack 0. • Low byte is the slot number of network module in rack 0. • High byte can be used to extend the number of sockets. • 00: Provided for full backward compatibility with applications created on firmware version 3.3 or earlier. • 01: Up to 64 sockets can be used (firmware version must be higher than 3.3).
Mask_Index	INT	Index of first word in the <code>Socket_Activity</code> array
Gest_Index	INT	Index of first word in <code>Management_Param</code> array

The following table describes the input/output parameters:

Parameter	Type	Comment
Management_Param	ARRAY [0... 3] OF INT	Function management array (see page 34) For this function, the operation report always returns the value 16#00 indicating that no error has occurred.

The following table describes the output parameters:

Parameter	Type	Comment
Socket_Activity	ARRAY [0... 1] OF INT - or - ARRAY [0... 3] OF INT (Refer to the note in the Function Description (see page 75) topic.)	Status of each socket. Each bit set to 1 indicates an event on the socket which corresponds to this bit. For example: For version 3.3 or earlier: ● If bit 5 of the first word has the value 1, socket 6 will be read by the <code>FCT_ACCEPT</code> function. ● If bit 3 of the second word has the value 1, socket 20 will be read by the <code>FCT_RECEIVE</code> function. For versions later than 3.3: ● If a bit is set to 1 on a listening socket, a server is ready to accept this connection. If a bit is set to 1 on a connected socket, data is ready to be sent or communication has been interrupted. Client Perspective: Once you have a client connection, the corresponding bit for the connection (socket number) is set to 1 in the array. Example: Socket 33 will be found in the third word, first bit (bit 0). Server Perspective: The listening function opens a socket. This socket number has a corresponding bit set to 1 in the array. Example: Socket 3 will be found in the first word, third bit (bit 2). Then, the Accept function will return a different socket number, and this socket number will have a corresponding bit set to 1 in the array. NOTE: The listening function socket number bit will remain set to 1 until the socket is closed.

Chapter 14

FCT_SEND: Sending Data to a Specified Socket

Description

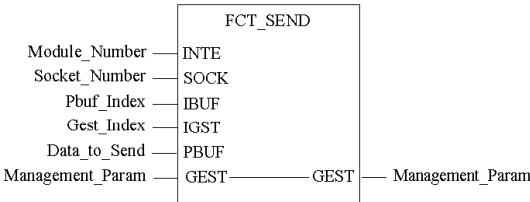
Function Description

The `FCT_SEND` function is used to send data to a foreign socket. The maximum length of data to send is 240 bytes.

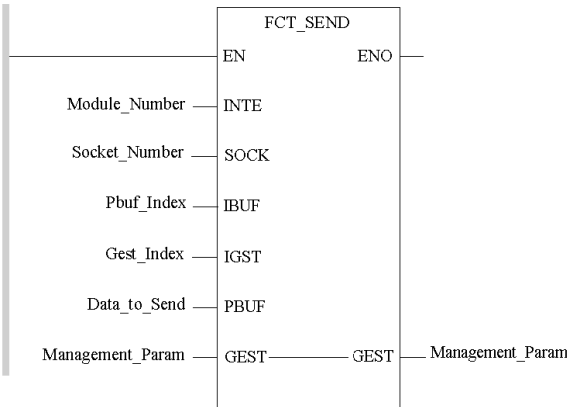
NOTE: it is not possible to send out of band data.

The additional parameters `EN` and `ENO` may be configured.

Representation in FBD



Representation in LD



Representation in IL

```
LD Module_Number
```

```
FCT_SEND Socket_Number, Pbuf_Index, Gest_Index, Data_to_Send, Management_Param
```

Representation in ST

```
FCT_SEND(Module_Number, Socket_Number, Pbuf_Index, Gest_Index, Data_to_Send, Management_Param);
```

Description of Parameters

The following table describes the input parameters:

Parameter	Type	Comment
Module_Number	INT	Slot number of network module in rack 0. ● Low byte is the slot number of network module in rack 0. ● High byte can be used to extend the number of sockets. ● 00: Provided for full backward compatibility with applications created on firmware version 3.3 or earlier. ● 01: Up to 64 sockets can be used (firmware version must be higher than 3.3).
Socket_Number	INT	Socket number
Pbuf_Index	INT	Index of first word in Pbuf_Address array
Gest_Index	INT	Index of first word in Management_Param array
Data_to_Send	ARRAY [0... n] OF INT	Array of a maximum of 120 words containing the data to be sent

The following table describes the input/output parameters:

Parameter	Type	Comment
Management_Param	ARRAY [0... 3] OF INT	Function management array (see page 34) The operation report can have the following values: ● 16#00: No error. ● 16#09: The socket number is invalid. ● 16#20: Communication is broken. ● 16#23: The socket is full. ● 16#36: The connection has been reset by peer. ● 16#39: The socket is not connected (listening socket). ● 16#0E: The length of the character string to be sent is greater than 240 bytes. The fourth word of the array should contain the number of bytes sent if no error has occurred.

Chapter 15

FCT_SETSOCKOPT: Sets the Options Associated with the Socket

Description

Function Description

The `FCT_SETSOCKOPT` function sets options associated with the specified socket. Some options are set automatically when the socket is created by the `FCT_SOCKET` ([see page 91](#)) function.

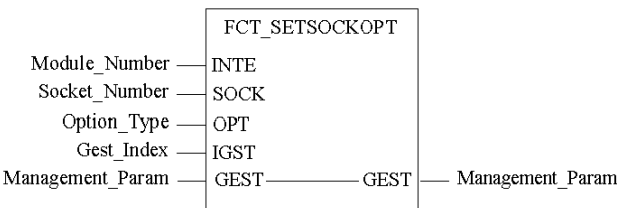
The following options are available:

- `DONT_ROUTE`: indicates that the outgoing data should not be routed. Packets directed at unconnected nodes are dropped.
- `RESET_DONT_ROUTE`: resets `DONT_ROUTE`.
- `KEEP_ALIVE`: ensures a connection is kept active by regularly and automatically sending packets on the socket.
- `RESET_KEEP_ALIVE`: resets `KEEP_ALIVE`.

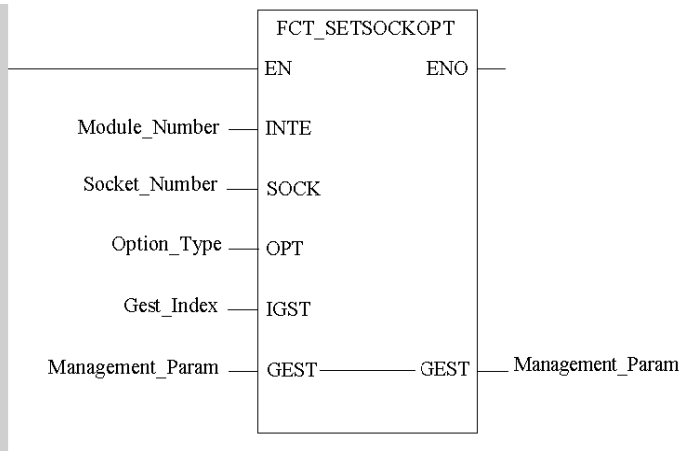
These options are selected by assigning a number in the `Option_Type` variable.

The additional parameters `EN` and `ENO` may be configured.

Representation in FBD



Representation in LD



Representation in IL

```
LD Module_Number
FCT_SETSOCKOPT Socket_Number, Option_Type, Gest_Index, Management_Param
```

Representation in ST

```
FCT_SETSOCKOPT(Module_Number, Socket_Number, Option_Type, Gest_Index, M
anagement_Param);
```

Description of Parameters

The following table describes the input parameters:

Parameter	Type	Comment
Module_Number	INT	Slot number of network module in rack 0. ● Low byte is the slot number of network module in rack 0. ● High byte can be used to extend the number of sockets. ● 00: Provided for full backward compatibility with applications created on firmware version 3.3 or earlier. ● 01: Up to 64 sockets can be used (firmware version must be higher than 3.3).
Socket_Number	INT	Socket number
Option_Type	INT	Type of option to be associated with socket. The values which can be assigned to this word are as follows: ● 1 for DONT_ROUTE ● 2 for RESET_DONT_ROUTE ● 3 for KEEP_ALIVE ● 4 for RESET_KEEP_ALIVE
Gest_Index	INT	Index of first word in <code>Management_Param</code> array

The following table describes the input/output parameters:

Parameter	Type	Comment
Management_Param	ARRAY [0... 3] OF INT	Function management array (see page 34) The operation report can have the following values: ● 16#00: No error. ● 16#09: The socket number is invalid. ● 16#16: Invalid option. The fourth word of the array should contain the number of bytes stored in the buffer.

Chapter 16

FCT_SHUTDOWN: Disables Transmission on the Socket

Description

Function Description

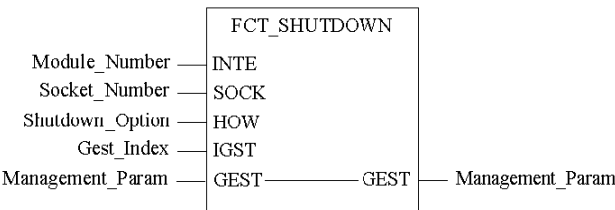
The `FCT_SHUTDOWN` function is used to disable either send/receive transmission on the socket.

The TCP window is not changed and incoming data will be accepted (but not acknowledged) until the window is exhausted.

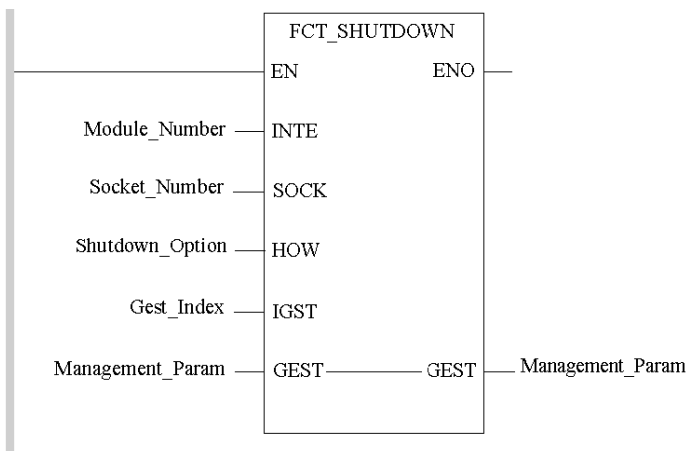
NOTE: the function does not close the socket, and resources assigned to the socket are not freed until the `FCT_CLOSE` is sent. However you should not attempt to reuse the socket after the `FCT_SHUTDOWN` function has been executed.

The additional parameters `EN` and `ENO` may be configured.

Representation in FBD



Representation in LD



Representation in IL

```
LD Module_Number
FCT_SHUTDOWN Socket_Number, Shutdown_Option, Gest_Index, Management_Param
```

Representation in ST

```
FCT_SHUTDOWN(Module_Number, Socket_Number, Shutdown_Option, Gest_Index,
Management_Param);
```

Description of Parameters

The following table describes the input parameters:

Parameter	Type	Comment
Module_Number	INT	Slot number of network module in rack 0. - Low byte is the slot number of network module in rack 0. - High byte can be used to extend the number of sockets. 00: Provided for full backward compatibility with applications created on firmware version 3.3 or earlier. 01: Up to 64 sockets can be used (firmware version must be higher than 3.3).
Socket_Number	INT	Socket number
Shutdown_Option	INT	Disable transmission option: 0: No further receives are allowed on the socket. 1: No further sends are allowed on the socket; a FIN message is sent. 2: No further sends or receives are allowed on the socket. This option does the same as both of the previous options together.
Gest_Index	INT	Index of first word in Management_Param array

The following table describes the input/output parameters:

Parameter	Type	Comment
Management_Param	ARRAY [0... 3] OF INT	Function management array (see page 34) The operation report can have the following values: 16#00: No error. 16#09: The socket number is invalid. 16#16: An argument is not valid. 16#39: The socket is not connected.

Chapter 17

FCT_SOCKET: Creation of a New Socket

Description

Function Description

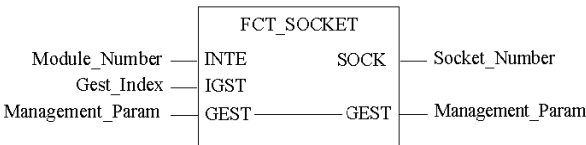
The `FCT_SOCKET` function creates a new socket and returns its socket number. The socket is a TCP/IP communication entity.

It is created as a `STREAM TCP` socket with the following options:

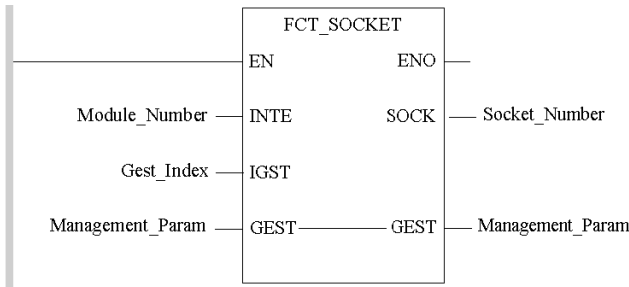
- `SO_LINGER` without timeout. This option controls the action taken when unsent data is queued on a socket and a `FCT_CLOSE` function is performed.
- `NO_DELAY`. The delay acknowledgment algorithm is disabled. Data is sent immediately over the network instead of waiting for the window to be completely full.
- `KEEP_ALIVE`. The connection is kept active by regularly and automatically sending packets on the socket.
- `REUSEADDR`. Authorizes the local port for reuse when a `FCT_BIND` is called.

The additional parameters `EN` and `ENO` may be configured.

Representation in FBD



Representation in LD



Representation in IL

```
LD Module_Number
```

```
FCT_SOCKET Gest_Index, Management_Param, Socket_Number
```

Representation in ST

```
FCT_SOCKET(Module_Number, Gest_Index, Management_Param, Socket_Number);
```

Description of Parameters

The following table describes the input parameters:

Parameter	Type	Comment
Module_Number	INT	Slot number of network module in rack 0. ● Low byte is the slot number of network module in rack 0. ● High byte can be used to extend the number of sockets. ● 00: Provided for full backward compatibility with applications created on firmware version 3.3 or earlier. ● 01: Up to 64 sockets can be assigned (firmware version must be higher than 3.3).
Gest_Index	INT	Index of first word in Management_Param array

The following table describes the output parameters:

Parameter	Type	Comment
Socket_Number	INT	Number of socket created if no error has occurred.

The following table describes the input/output parameters:

Parameter	Type	Comment
Management_Param	ARRAY [0... 3] OF INT	Function management array (see page 34) The operation report can have the following values: ● 16#00: No error. ● 16#37: The maximum number of sockets has been reached.

Part IV

DFB Library

Chapter 18

DFB Library

Overview

The TCP communication DFB library for Premium PLCs is used to transfer blocks of data between a PLC application and a remote application via a TCP/IP connection established at the initiative of the remote application.

The remote application is executed on a device that supports the TCP/IP communication profile.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
DFB Library Introduction	96
TCP_CNX DFB	97
TCP_SEND DFB	100
TCP_RECEIVE DFB	103

DFB Library Introduction

Overview

The communication DFB library is comprised of the following:

- connection management TCP_CNX DFB
- TCP_SEND DFB for transmission of data blocks (8 kb maximum size)
- TCP_RECEIVE DFB for receipt of data blocks (8 kb maximum size)

NOTE: TCP communication DFBs for Premium PLCs use the services from the *TCP Open for Premium* range to manage TCP connections and to send and receive byte flows on its connections.

The function blocks use the messaging services implemented in the Premium processor. As a result, all the communication blocks can be executed over several PLC cycles. The CPU/module exchange of a datagram is made per PLC cycle.

DFB Maintenance

The execution of function blocks is explicitly maintained. You must program maintenance of the DFB as long as its `ACTIVITY` bit is active.

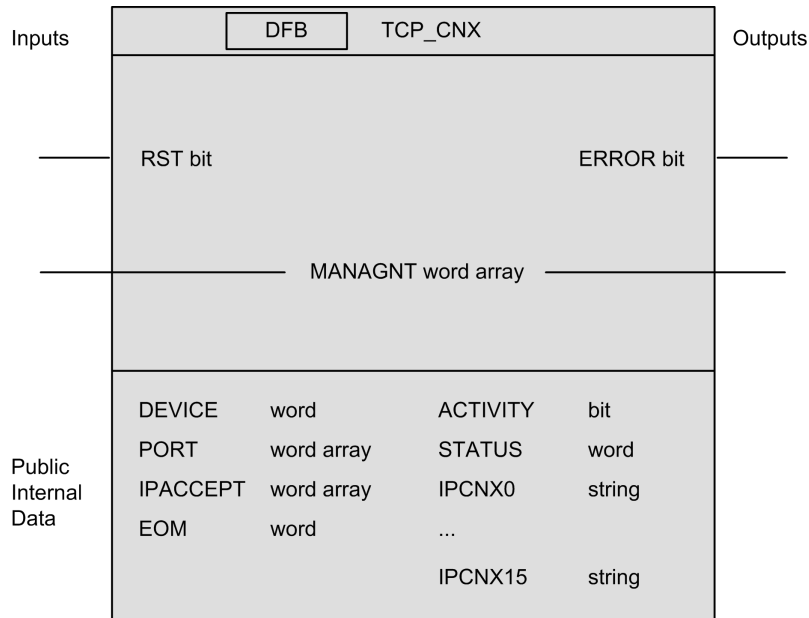
NOTE: Do not execute the same DFB instance more than once in the same PLC cycle.

Maintenance of the DFBs is mandatory since all the communication function blocks are executed over several PLC cycles. As soon as the DFB is called, it is restarted for each turn of the PLC cycle as long as its `ACTIVITY` bit is equal to 1.

TCP_CNX DFB

Overview

The TCP_CNX DFB manages the connections with remote clients as well as the operating modes of the connections. A TCP_CNX DFB is informed of breaks in connections by a TCP_SEND or TCP_RECEIVE DFB via the status of the management words. Only one TCP_CNX DFB instance exists for a TSX ETY 110 WS/5103 module.



Input Parameters

Parameter	Type	Description
RST	bit	Setting this input to 1: - interrupts the exchanges in progress - closes all connections

Input/Output Parameters

Parameter	Type	Description
MANAGNT	word table	Management table common to all DBFs

Output Parameters

Parameter	Type	Description
ERROR	bit	This output bit is set to 1 if the exchange is not carried out correctly. The status word indicates the type of error that has occurred.

Internal Public Data

Parameter	Type	Write/Read Variables	Description
DEVICE	word	W	Module number. Number of the physical slot where the TSX ETY 110 ES/5103 module is located in the rack.
PORT	word table	W	Local port number No. <i>i</i> . Table of 16 local service ports to be listened to (signed value ≥ 5010). The value 0 indicates a non-significant value.
IPACCEPT	double word	W	IP address of the remote machine No. <i>i</i> . Word 4-byte IP address table from table 8. Remote machines authorized to connect to the PLC. The value 9 indicates a non-significant value.
EOM	byte	W	End of Message character. This parameter defines the presence and the value of an End of Message character to be used in the message exchanges: • EOM = 0: no End of Message character • EOM from 0x01 to 0xFF: value of End of Message character: • added by the module during transmission • checked and deleted by the module during reception
IPCNXi	character string	R	IP address of the remote machine No. <i>i</i> . Read-only character string to be used for maintenance only. Indicates the IP address of the remote machine in the format <i>aaa.bbb.ccc.ddd</i> connected to the port with index <i>i</i> in the configuration. The value 0.0.0.0 indicates that no machine is connected to this port.

Parameter	Type	Write/Read Variables	Description
ACTIVITY	bit	R	This output bit is at 1 when the DFB is in progress and needs to be maintained. It is set to 0 with a warm or cold restart or a DFB RESET.
STATUS	word	R	This word is only meaningful if the ERROR output bit is set to 1. It indicates the code of the error that occurred during the exchange. (Each word bit set to 1 indicates an error.)

Operation

The TCP_CNX DFB must be called at each turn of the PLC cycle to ensure permanent management of the TCP ports.

Listening to the configured ports starts if the `RST` and `ACTIVITY` bits are set to 0. Then, the `ACTIVITY` bit remains at 1 until the application allows the `RST` bit to return to 0.

If communication with the module is not correct, the output `ERROR` bit is set to 1.

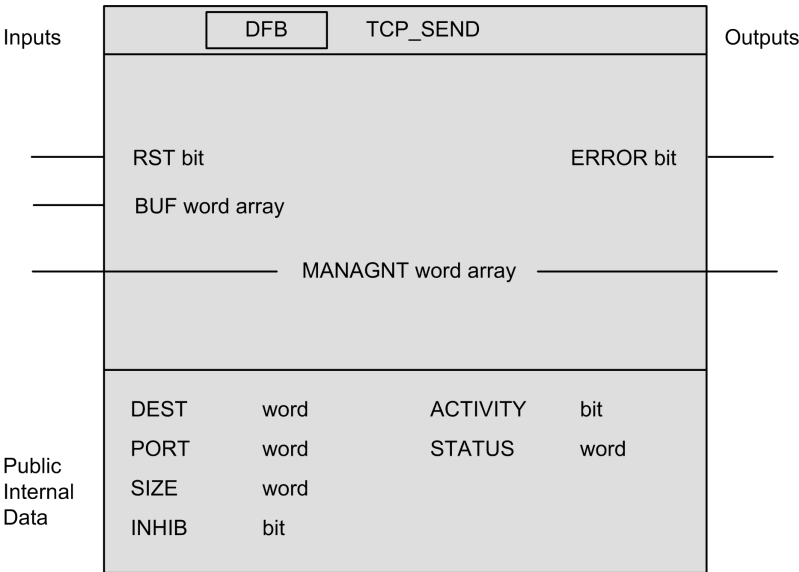
At any time, setting the `RST` bit (priority input) to 1 enables all the connections in progress or on standby to be interrupted. The `ACTIVITY` bit (exchange finished) is set to 0, and the `ERROR` bit is set to 1. The status word `<TCP_CNX instance name>.STATUS` indicates the type of error.

At a cold or warm restart, the TCP_CNX DFB automatically returns to listen mode for the configured ports.

TCP_SEND DFB

Overview

The TCP_SEND DFB allows a data message to be sent to a remote client application via a TCP/IP connection. The maximum size of the message is 8 kb.



Input Parameters

Parameter	Type	Description
RST	bit	Setting this bit to 1: ● interrupts the exchange in progress (if the ACTIVITY bit is set to 1) ● closes the connection ● sets the ACTIVITY information bit to 0 and the ERROR output bit to 1 The error code is found in the STATUS word.
BUF	word table	This variable defines the address of the first %MWi word of the buffer in order to send the message.

Input/Output Parameters

Parameter	Type	Description
MANAGNT	word table	Management table common to all DFBs

Output Parameters

Parameter	Type	Description
ERROR	bit	This output bit is set to 1 if the exchange is not carried out correctly. The <code>STATUS</code> word indicates the type of error that has occurred.

Internal Public Data

Parameter	Type	Write/Read Variables	Description
DEST	double word	W	This variable defines the IP address of the remote machine on which the client is connected to the local port. By default, <code>DEST = 0</code> . The message is sent to the only remote client station connected to the local service port of the PLC.
PORT	word	W	This variable defines the local port number to which the remote client is connected.
SIZE	word	W	This variable defines the size of the message to be sent, from 0 to 8192 bytes.
ACTIVITY	bit	R	This bit is set to 1 when an exchange is in progress. It is set to 0 when the exchange is finished.
INHIB	bit	R	This bit enables the error warning to be inhibited. The <code>ERROR</code> output bit and the <code>STATUS</code> word remain at 0 (execution of the block is not interrupted).
STATUS	word	R	This word is only meaningful if the <code>ERROR</code> output bit is set to 1. It indicates the error code that occurred during the exchange (each word bit set to 1 indicates an error).

Operation

Data transfer is triggered when the TCP_SEND DFB is called, if the RST and ACTIVITY bits are set to 0. During the exchange, the ACTIVITY bit is set to 1. At the end of transmission, the ACTIVITY bit is set to 0. In addition, if the exchange is not correct, the ERROR output bit is set to 1.

At any time, setting the RST bit (priority input) to 1 allows the exchange in progress to be interrupted. The ACTIVITY bit (exchange finished) is set to 0, and the ERROR bit is set to 1. The status word <TCP_SEND instance name>.STATUS indicates the type of error. If the connection is open, it closes, and the module waits for the incoming connection. The TCP_CNX DFB ([see page 97](#)) reestablishes the connection to the TCP service port to start sending messages again.

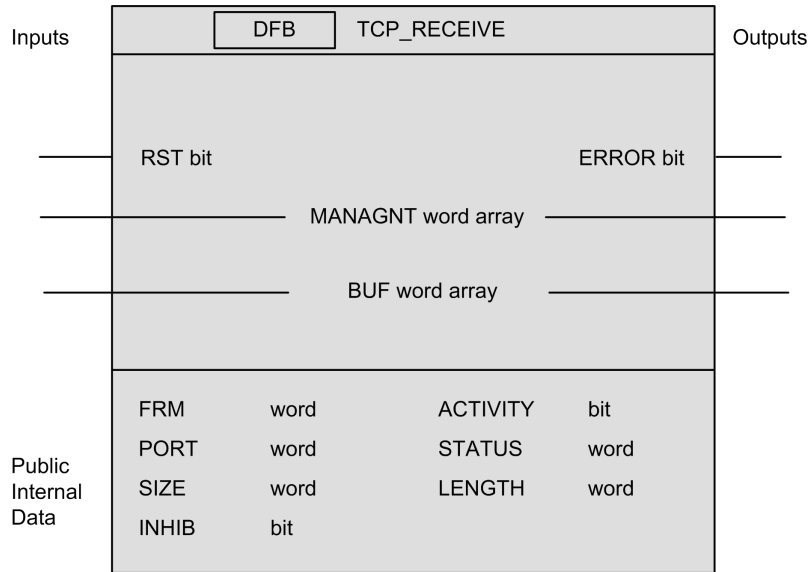
When a message is sent, the following process occurs:

- check input parameters (IP address of the remote machine, number of local port)
- search for the open connection with the remote client requested
- send message by block of 240 bytes (adding the End of Message character, if the option is requested)

TCP_RECEIVE DFB

Overview

The TCP_RECEIVE DFB allows a data message to be received from a remote client application via a TCP/IP connection. The maximum size of the message is 8 kb.



Input Parameters

Parameter	Type	Description
RST	bit	Setting this bit to 1: - interrupts the exchange in progress (if the ACTIVITY bit is set to 1) - closes the connection - sets the ACTIVITY bit to 0 and the ERROR output bit to 1 The error code is found in the STATUS word.

Input/Output Parameters

Parameter	Type	Description
BUF	word table	This input and output parameter defines the address of the first %MWi word of the buffer in order to receive the message.

Parameter	Type	Description
MANAGNT	word table	Management table common to all DFBs

Output Parameters

Parameter	Type	Description
ERROR	bit	This output bit is set to 1 if the exchange is not carried out correctly. The <code>STATUS</code> word indicates the type of error that has occurred.

Internal Public Data

Parameter	Type	Write/Read Variables	Description
FRM	double word	W	This variable defines the IP address of the remote machine on which the client is connected. By default, <code>FRM</code> = 0, and the DFB awaits the message sent by the unique remote client connected to the local service port of the PLC.
PORT	word	W	This variable defines the number of the local port to which the remote client is connected.
SIZE	word	W	This variable defines the size of the message to be sent, from 0 to 8192 bytes.
ACTIVITY	bit	R	This variable is set to 0 by the DFB when the exchange is finished. If the <code>ERROR</code> output bit is set to 0, the <code>ACTIVITY</code> bit indicates that the message has been received successfully. However, if the <code>ERROR</code> bit is set to 1, the <code>ACTIVITY</code> bit indicates that the exchange finished, but has errors.
INHIB	bit	W	This variable allows the error warning to be inhibited. The <code>ERROR</code> output bit and the <code>STATUS</code> word remain at 0 (execution of the block is not interrupted).
STATUS	word	R	This word is only significant if the <code>ERROR</code> output bit is set to 1. It indicates the error code that occurred during the exchange (each word bit set to 1 indicates an error).
LENGTH	word	R	This variable contains the number of bytes received if the <code>ERROR</code> output bit is set to 0.

Operation

Data transfer is triggered when the `TCP_RECEIVE` DFB is called, if the `RST` and `ACTIVITY` bits are set to 0. During the exchange, the `ACTIVITY` bit is set to 1. At the end of transmission, the `ACTIVITY` bit is set to 0. In addition, if the exchange is not correct, the `ERROR` output bit is set to 1.

At any time, setting the `RST` bit (priority input) to 1 allows the exchange in progress to be interrupted. The `ACTIVITY` bit (exchange finished) is set to 0, and the `ERROR` bit is set to 1. The status word `<TCP_RECEIVE instance name>.STATUS` indicates the type of error. If the connection is open, it closes, and the module waits for the incoming connection. Any messages in the module's receive buffers are lost.

When a message is received, the following process occurs:

- check input parameters (IP address of the remote machine, number of local port)
- search for the open connection with remote client requested
- EF block receiving the message is queued to receive message

As soon as a DFB is active for a given connection, the module awaits the message to be received, and the first two bytes received indicate the length of the message. The data is then transferred into the PLC's words in frames of 240 bytes.

When the full message is copied onto the PLC:

- The DFB checks the end of character message (if this option has been configured).
- The `STATUS` and `LENGTH` parameters of the DFB are filled in, and the `ACTIVITY` and `ERROR` bits are set for the module application.



E

EN

EN stands for **EN**able; it is an optional block input. When the **EN** input is enabled, an **ENO** output is set automatically.

If **EN** = 0, the block is not enabled; its internal program is not executed, and **ENO** is set to 0.

If **EN** = 1, the block's internal program is run and **ENO** is set to 1. If an error occurs, **ENO** is set to 0.

If the **EN** input is not connected, it is set automatically to 1.

ENO

ENO stands for **Error NO**tification; this is the output associated with the optional input **EN**.

If **ENO** is set to 0 (because **EN** = 0 or in case of an execution error):

- the status of the function block outputs remains the same as it was during the previous scanning cycle that executed correctly;
- the output(s) of the function, as well as the procedures, are set to "0".

I

INT

INT is the abbreviation of single **INT**eger (encoded in 16 bits).

The upper/lower limits are as follows: $-(2 \text{ to the power of } 15)$ to $(2 \text{ to the power of } 15) - 1$.

Example:

-32768, 32767, 2#11111110001001001, 16#9FA4.



A

availability of the instructions, 21

D

DFB, 95

F

FCT_ACCEPT, 53, 53
FCT_BIND, 57, 57
FCT_CLOSE, 61, 61
FCT_CONNECT, 65, 65
FCT_LISTEN, 69, 69
FCT_RECEIVE, 71, 71
FCT_SELECT, 75, 75
FCT_SEND, 79, 79
FCT_SETSOCKOPT, 83, 83
FCT_SHUTDOWN, 87, 87
FCT_SOCKET, 91, 91

I

instructions
 availability, 21

T

TCP Open - instructions
 FCT_ACCEPT, 53
 FCT_BIND, 57
 FCT_CLOSE, 61
 FCT_CONNECT, 65
 FCT_LISTEN, 69
 FCT_RECEIVE, 71
 FCT_SELECT, 75
 FCT_SEND, 79
 FCT_SETSOCKOPT, 83
 FCT_SHUTDOWN, 87
 FCT_SOCKET, 91
 general information, 51
TCP_CNX, 95
TCP_RECEIVE, 95
TCP_SEND, 95

