

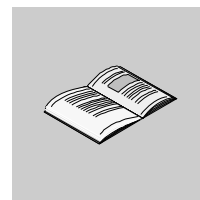
# Unity Pro 2.3 TCP Open Block Library

33002551



---

# Table of Contents



---

<b>About the Book</b> .....	<b>5</b>
<b>Part I General</b> .....	<b>7</b>
<b>Chapter 1 Block types and their applications</b> .....	<b>9</b>
Block types .....	10
FFB Structure .....	11
EN and ENO .....	14
<b>Chapter 2 Availability of the blocks on the various hardware platforms</b>	<b>17</b>
<b>Part II Introduction to TCP Open</b> .....	<b>19</b>
<b>Chapter 3 General</b> .....	<b>21</b>
<b>Chapter 4 Warnings</b> .....	<b>23</b>
<b>Chapter 5 Description of Operation</b> .....	<b>25</b>
Operating Rules for TCP Open EFs .....	26
Principle Governing Socket Operation .....	27
General Structure of a TCP Open Communication EF .....	29
Structure of TCP Open management parameters .....	30
Management parameters: communication and operation reports .....	31
The client/server model .....	33
Example of client/server applications .....	35
<b>Chapter 6 Operating modes and performance</b> .....	<b>39</b>
Operating modes of the network module .....	40
Performance .....	41
Debugging and diagnostics .....	42

---

<b>Part III</b>	<b>Advanced</b>	<b>43</b>
Chapter 7	FCT_ACCEPT: Accepts a connection request	45
Chapter 8	FCT_BIND: Binds a socket number to an IP address and a port	49
Chapter 9	FCT_CLOSE: Deletes the specified socket	53
Chapter 10	FCT_CONNECT: Establishes a connection with an IP address	57
Chapter 11	FCT_LISTEN: Configuration of a socket await connection	61
Chapter 12	FCT_RECEIVE: Retrieves data available on a socket	63
Chapter 13	FCT_SELECT: Multiplexes requests over sockets	67
Chapter 14	FCT_SEND: Sending data to a specified socket	71
Chapter 15	FCT_SETSOCKOPT: Sets the options associated with the socket	75
Chapter 16	FCT_SHUTDOWN: Disables transmission on the socket	79
Chapter 17	FCT_SOCKET: Creation of a new socket	83
<b>Appendices</b>		<b>87</b>
<b>Appendix A</b>	<b>System objects</b>	<b>89</b>
	System bit introduction	90
	Description of system bits %S15 to %S21	91
	Description of System Words %SW12 to %SW19	94
<b>Glossary</b>		<b>97</b>
<b>Index</b>		<b>113</b>

# About the Book

## At a Glance

---

**Document Scope** This document describes the functions and function blocks of the TCP Open library. This document is valid for Unity Pro Version 2.3.

---

**Validity Note** The data and illustrations found in this document are not binding. We reserve the right to modify our products in line with our policy of continuous product development. The information in this document is subject to change without notice and should not be construed as a commitment by Schneider Electric.

---

**Product Related Warnings** Schneider Electric assumes no responsibility for any errors that may appear in this document. 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 ensure compliance with documented system data, only the manufacturer should perform repairs to components.

When controllers are used for applications with technical safety requirements, please follow the relevant instructions.

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 product related warning can result in injury or equipment damage.

---

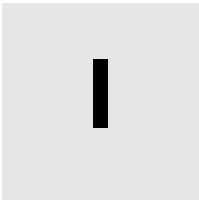
**User Comments** We welcome your comments about this document. You can reach us by e-mail at [techpub@schneider-electric.com](mailto:techpub@schneider-electric.com)

---



---

# General



---

## Introduction

**Overview** The following chapters contain general information about the TCP Open library.

**What's in this Part?** This part contains the following chapters:

Chapter	Chapter Name	Page
1	Block types and their applications	9
2	Availability of the blocks on the various hardware platforms	17



---

# Block types and their applications

# 1

---

## Introduction

### Overview

This chapter describes the different block types and their applications.

### What's in this Chapter?

This chapter contains the following topics:

Topic	Page
Block types	10
FFB Structure	11
EN and ENO	14

## Block types

---

### Block types

Different block types are used in Unity Pro. The general term for all block types is FFB. There are the following types of block:

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

### Elementary Function

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

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

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

---

### Elementary function block

Elementary function blocks (EFB) have an internal status. If the inputs have the same values, the value on the output 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 (FDB and LD) as a block frame with inputs and outputs. The inputs are always represented on the left and the outputs always on the right of the frame. The name of the function block, i.e. the function block type, is shown in the center of the frame. The instance name is displayed above the 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 technical functions.

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.

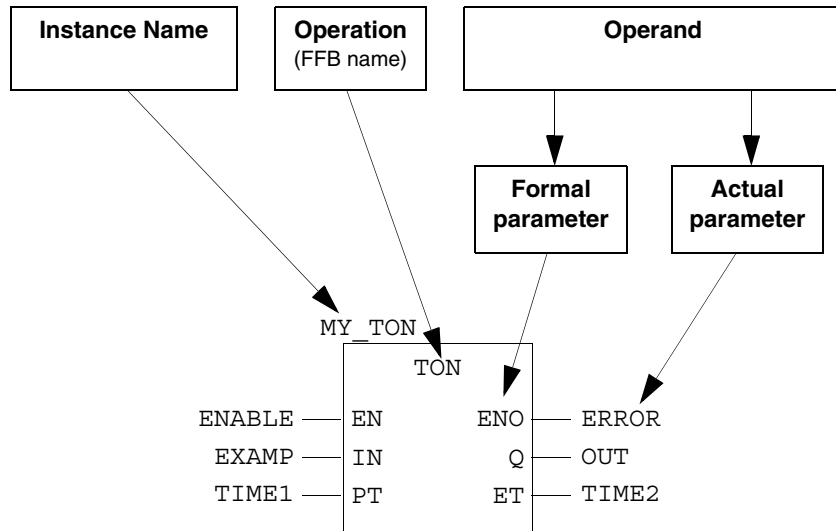
---

## FFB Structure

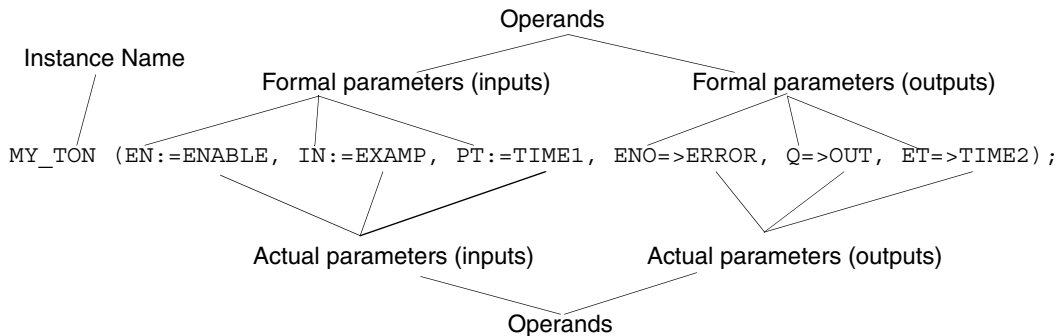
### Structure

Each FFB is made up of an operation (name of the FFB), the operands 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:



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

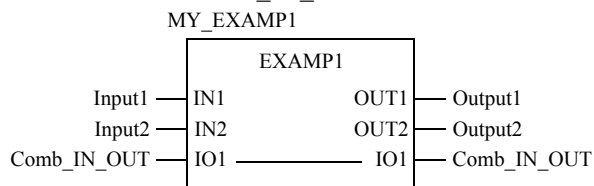


### Operation

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

<b>Operand</b>	The operand specifies what the operation is to be executed with. With FFBs, this consists of formal and actual parameters.
<b>Formal/actual parameters</b>	<p>Inputs and outputs are required to transfer values to or from an FFB. These are called formal parameters.</p> <p>Objects are linked to formal parameters; these objects contain the current process states. They are called actual parameters.</p> <p>At program runtime, the values from the process are transferred to the FFB via the actual parameters and then output again after processing.</p> <p>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 all actual parameters consist of literals, a suitable data type is selected for the function block.</p>
<b>FFB Call in IL/ST</b>	<p>In text languages IL and ST, FFBs can be called in formal and in informal form. Details can be found in the <i>Reference manual</i>.</p> <p>Example of a formal function call:</p> <pre>out:=LIMIT (MN:=0, IN:=var1, MX:=5) ;</pre> <p>Example of an informal function call:</p> <pre>out:=LIMIT (0, var1, 5) ;</pre> <div><b>Note:</b> Take note that the use of <code>EN</code> and <code>ENO</code> is only possible for formal calls.</div>
<b>VAR_IN_OUT variable</b>	<p>FFBs are often used to read a variable at an input (input variables), to process it and to output the altered values of the <b>same</b> variable (output variables).</p> <p>This special type of input/output variable is also called a <code>VAR_IN_OUT</code> variable.</p> <p>The input and output variable are linked in the graphic languages (FBD and LD) using a line showing that they belong together.</p>

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:

- All 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 at 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

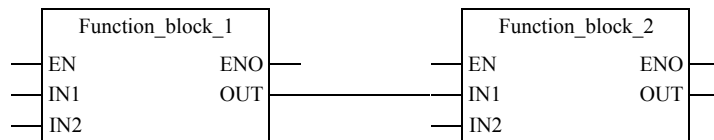
An **EN** input and an **ENO** output can be configured for all FFBs.

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

If the value of **EN** is 1 when the FFB is called up, the algorithms defined by the FFB are executed. After the algorithms have been executed successfully, the value of **ENO** is set to 1. If an error occurs when executing these algorithms, **ENO** is set to 0.

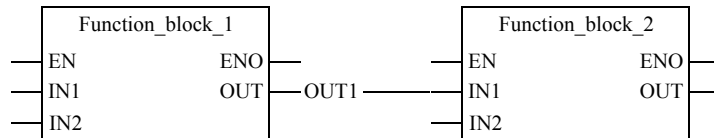
If **ENO** is set to 0 (caused by **EN**=0 or an error during execution):

- 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:



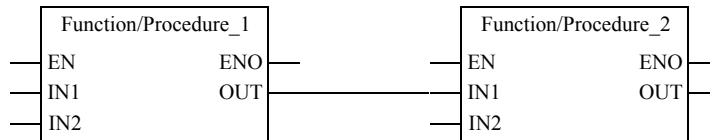
If **EN** from **FunctionBlock\_1** is set to 0, the output connection **OUT** from **FunctionBlock\_1** retains the status it had in the last correctly executed cycle. The variable **OUT1** on the same pin, either retains its previous status or can be changed externally without influencing the connection. The variable and the link are saved independently of each other.

- Functions/Procedures
 

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

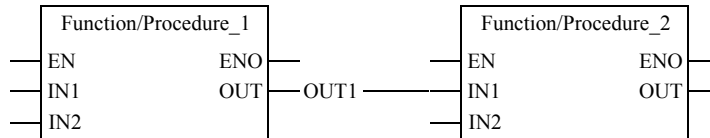
Here nevertheless an explanation of the output status in this case:

- 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 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 Function/Procedure\_1 is set to 0, the output connection OUT from Function/Procedure\_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.

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 always processed)

### 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.



---

# Availability of the blocks on the 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.

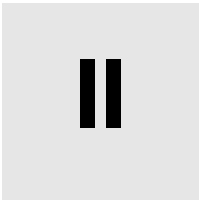
**Advanced** Availability of the blocks:

Block name	Block type	Premium	Quantum
FCT_ACCEPT	Procedure	+	-
FCT_BIND	Procedure	+	-
FCT_CLOSE	Procedure	+	-
FCT_CONNECT	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		



---

# Introduction to TCP Open



---

## At a Glance

**Topic of this Part** This part introduces the TCP Open service, which can be used with Elementary Functions (EF) in Unity Pro.

**What's in this Part?** This part contains the following chapters:

Chapter	Chapter Name	Page
3	General	21
4	Warnings	23
5	Description of Operation	25
6	Operating modes and performance	39



---

### 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

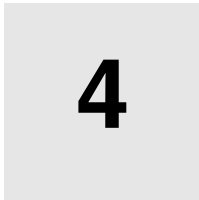
This documentation contains a description of the EFs for TCP Open; the documentation concerning the DFBs for TCP Open is supplied on the installation CD.

---



---


# Warnings



---

## Notes and Warnings

<b>At a Glance</b>	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.
<b>Observations</b>	<p>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.</p> <p>The implementation of Client/Server services follows certain programming rules which you must understand.</p>

 **WARNING**

**Prerequisites for implementation of TCP Open EFs.**

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 this instruction 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**

**Schneider Electric is not liable for:**

- The design and validation of the communication system architecture (client/ server operating modes and protocols, performances, etc.)
- The implementation of appropriate EFs (or reuse of example EFs included in the TCP Open kit)
- The testing and the validation of EFs integrated into the communication system architecture
- Maintenance and diagnostics errors

**Failure to follow this instruction can result in death, serious injury, or equipment damage.**

---

---

# Description of Operation

# 5

---

## At a Glance

### 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's in this Chapter?

This chapter contains the following topics:

Topic	Page
Operating Rules for TCP Open EFs	26
Principle Governing Socket Operation	27
General Structure of a TCP Open Communication EF	29
Structure of TCP Open management parameters	30
Management parameters: communication and operation reports	31
The client/server model	33
Example of client/server applications	35

## 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` function (see p. 83).

---

---

## 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 p. 83) function or the `FCT_ACCEPT` (see p. 45) 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 p. 49) function.
3	Set up the socket to accept connections using the <code>FCT_LISTEN</code> (see p. 61) function. <b>Note:</b> the PLC acts as TCP server, the initial socket listens, and receives client socket connections.
4	Apply the <code>FCT_ACCEPT</code> (see p. 45) function to this socket to create a new socket which will establish the connection.  <b>Note:</b> 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.

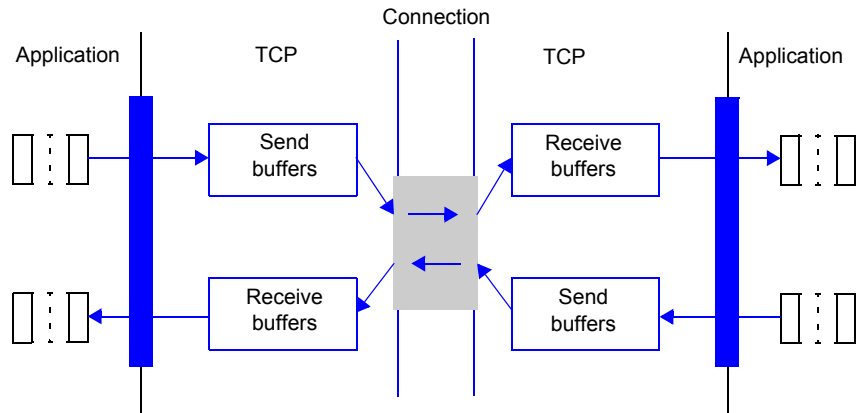
**Establishing a Client Connection**

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 p. 71) and `FCT_RECEIVE` (see p. 63) functions. The diagram below shows how these exchanges work.

**Managing Sockets**

Other functions can be used to act on sockets:

- `FCT_SETSOCKOPT` (see p. 75): associates options with a socket. These options modify the behavior of the socket.
- `FCT_SELECT` (see p. 67): used to test events on sockets.
- `FCT_SHUTDOWN` (see p. 79): disables transmission on the socket.
- `FCT_CLOSE` (see p. 53): 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:

- An 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 *p. 30*). 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.

### Reports

Operation reports are described in *Operation Report*, p. 32. Communication reports are described in *Communication Report*, p. 31.

### Length

The length parameter is used with the `FCT_SEND` (see p. 71) and `FCT_RECEIVE` (see p. 63) functions. The length parameter is also used with the `FCT_SELECT` (see p. 57) 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
16#05	Incorrect management parameter format
16#06	Incorrect specific parameters
16#07	Network module missing or incorrect address
16#0B	No processor system resources
16#0E	Incorrect send length
16#FF	Message refused

**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	Inadequate system resources (too many EFs in the same PLC cycle)
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.

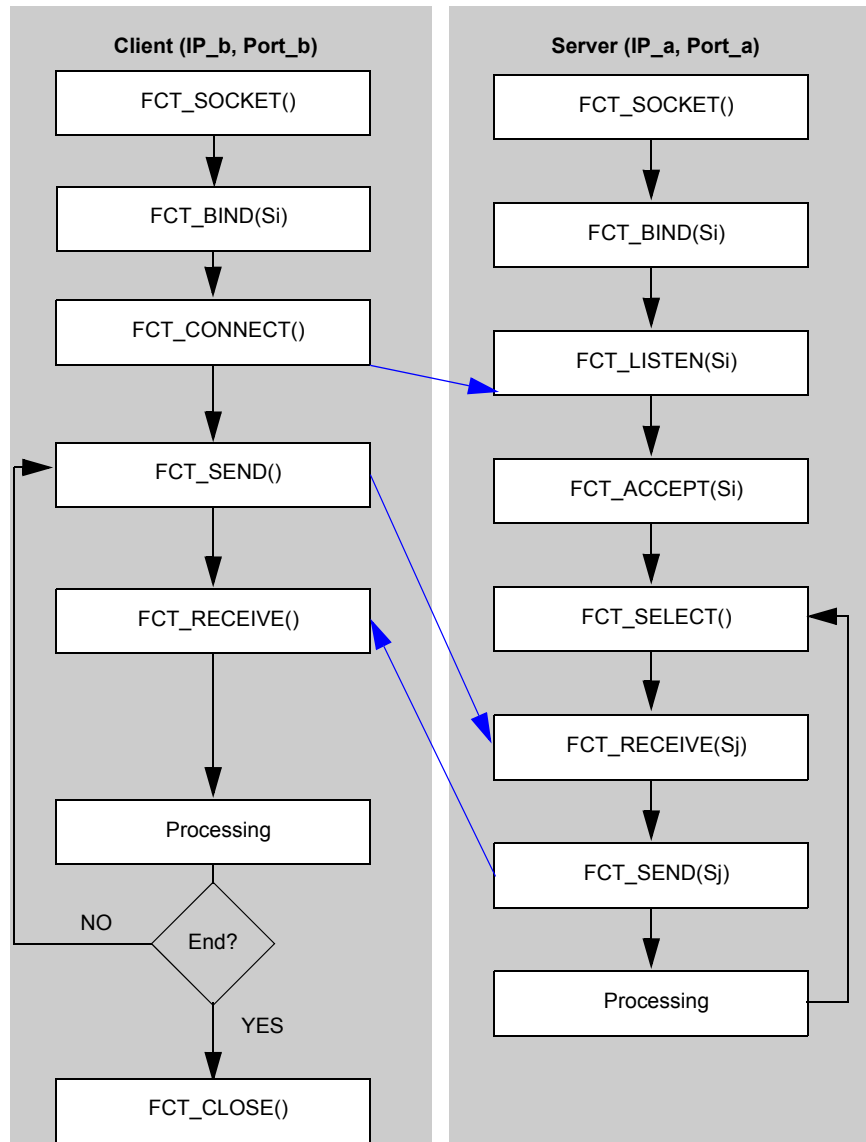
The client/server model operates according to a specific protocol which must be implemented at either end of the connection. This protocol is described in the following paragraphs.

**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.
-

**The Client/Server Model**

**S<sub>i</sub>** is assigned to the address: IP<sub>a</sub>, Port<sub>a</sub>

**S<sub>j</sub>** is assigned to the external socket with the address: IP<sub>b</sub>, Port<sub>b</sub>

## 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.

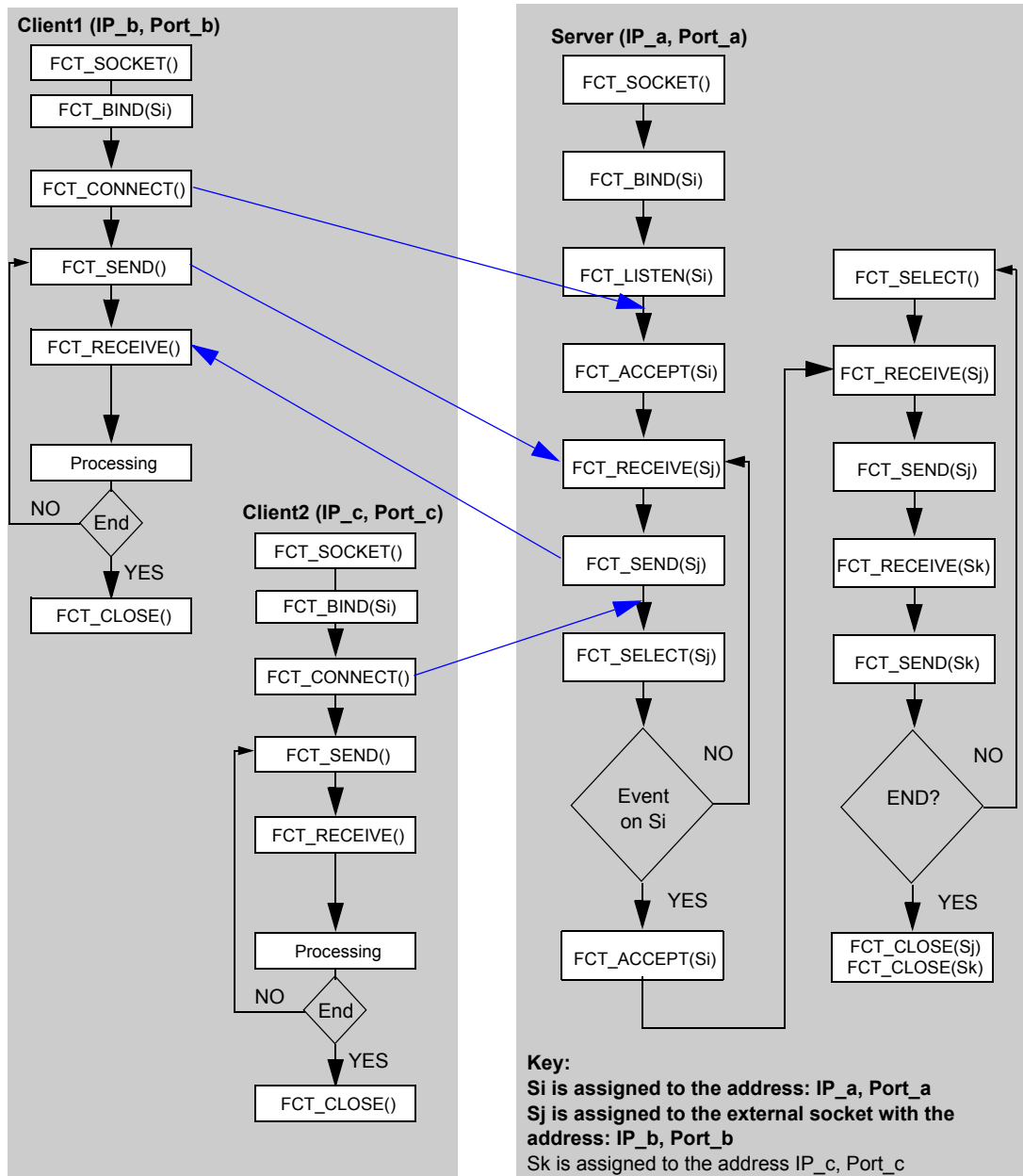
The client side is described in the form of a flow diagram, whereas on server side is shown as a sequence of operations closely linked to the events on the client side.

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.

---

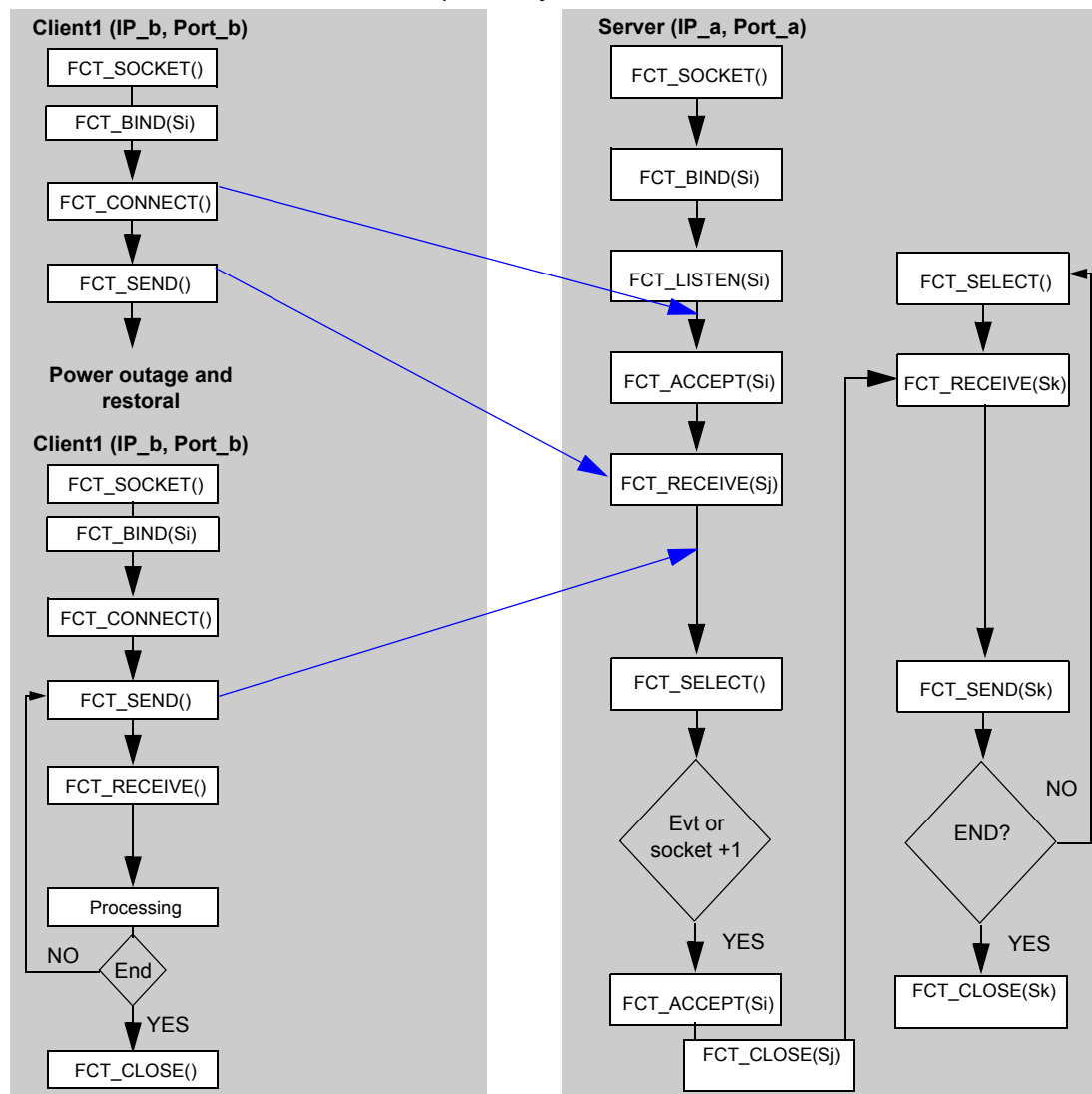
**Example 1**

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.



**Si is assigned to the address: IP\_a, Port\_a**

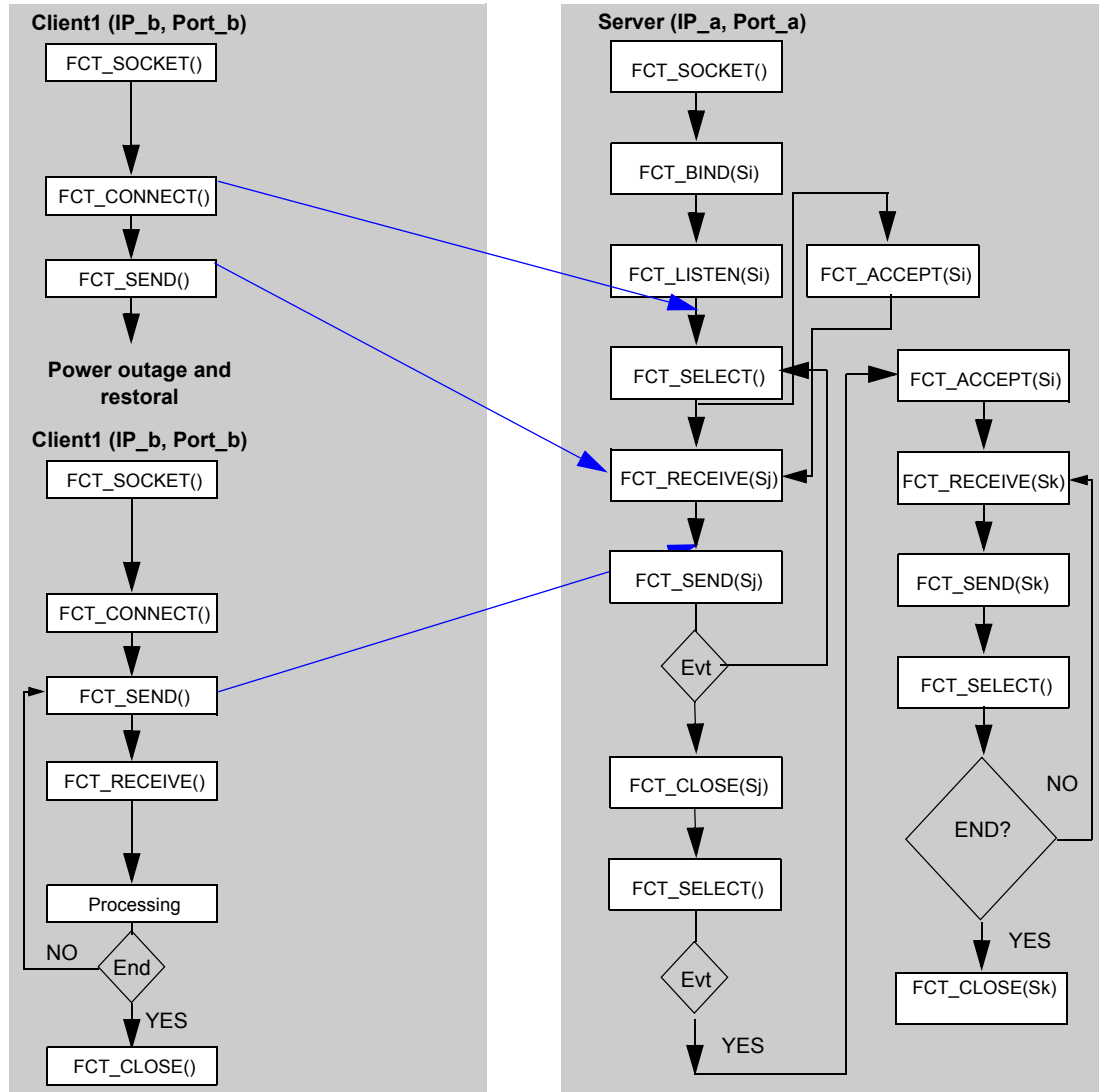
**Sj is assigned to the external socket with the address: IP\_b, Port\_b**

Sk is assigned to the address IP\_b, Port\_b, a new socket is created on receiving the second request from the client.

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.

**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 nothing has been sent by the server, the socket is still considered to be connected.



**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.

---

# Operating modes and performance

6

---

## At a Glance

### 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's in this Chapter?

This chapter contains the following topics:

Topic	Page
Operating modes of the network module	40
Performance	41
Debugging and diagnostics	42

## 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

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

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

#### **WARNING**

##### **Pay attention to restart management.**

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 this instruction can result in death, serious injury, or equipment damage.**

## 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.

<p><b>Note:</b> 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.</p>
--

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 *p. 71*) or FCT\_RECEIVE (see *p. 63*) 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.

---

---

# Advanced



---

## Introduction

**Overview** The following chapters describe the elementary functions and elementary function blocks of the `Advanced` family.

**What's in this Part?** This part contains the following chapters:

Chapter	Chapter Name	Page
7	FCT_ACCEPT: Accepts a connection request	45
8	FCT_BIND: Binds a socket number to an IP address and a port	49
9	FCT_CLOSE: Deletes the specified socket	53
10	FCT_CONNECT: Establishes a connection with an IP address	57
11	FCT_LISTEN: Configuration of a socket await connection	61
12	FCT_RECEIVE: Retrieves data available on a socket	63
13	FCT_SELECT: Multiplexes requests over sockets	67
14	FCT_SEND: Sending data to a specified socket	71
15	FCT_SETSOCKOPT: Sets the options associated with the socket	75
16	FCT_SHUTDOWN: Disables transmission on the socket	79
17	FCT_SOCKET: Creation of a new socket	83



---

# 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:

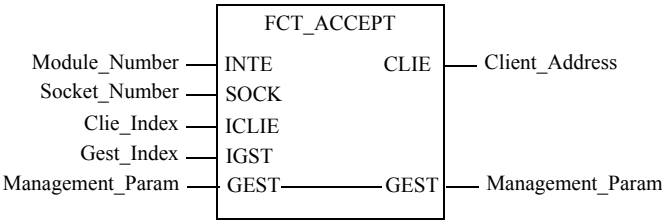
- extracts the first connection request in the queue of pending connections;
- creates a connected socket with the same properties as the original socket;
- completes the connection between the foreign socket and the new socket;
- and returns a number for the new socket.

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.

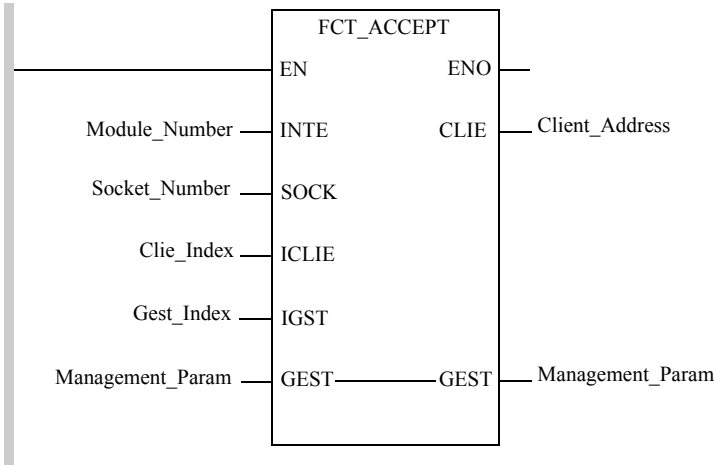
The additional parameters `EN` and `ENO` may be configured.

---

### 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);
```

**Parameters**

The following table describes the input parameters:

Parameter	Type	Comment
Module_Number	INT	Slot number of network module in rack 0. <ul style="list-style-type: none"> <li>● Low byte is the slot number of network module in rack 0</li> <li>● High byte can be used to extend the number of sockets <ul style="list-style-type: none"> <li>● 00 - provided for full backward compatibility with applications created on firmware version 3.3 or earlier</li> <li>● 01- up to 64 sockets can be used (firmware version must be higher than 3.3)</li> </ul> </li> </ul>
Socket_Number	INT	Socket number
Clie_Index	INT	Index of first word in <code>Client_Address</code> array
Gest_Index	INT	Index of first word in <code>Management_Param</code> 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: <ul style="list-style-type: none"> <li>● <code>Client_Address[0]</code>: connected socket number</li> <li>● <code>Client_Address[1]</code>: client port number</li> <li>● <code>Client_Address[2]</code>: least significant word of the client IP address</li> <li>● <code>Client_Address[3]</code>: most significant word of the client IP address</li> </ul>

The following table describes the input/output parameters:

Parameter	Type	Comment
Management_Param	ARRAY [0... 3] OF INT	Function management array (see <i>p. 30</i> ) The operation report can have the following values: <ul style="list-style-type: none"> <li>● 16#00: no error</li> <li>● 16#09: the socket number is invalid</li> <li>● 16#16: the <code>FCT_LISTEN</code> function must be called before <code>FCT_ACCEPT</code></li> <li>● 16#23: no connection request</li> </ul>



---

# FCT\_BIND: Binds a socket number to an IP address and a port



---

## Description

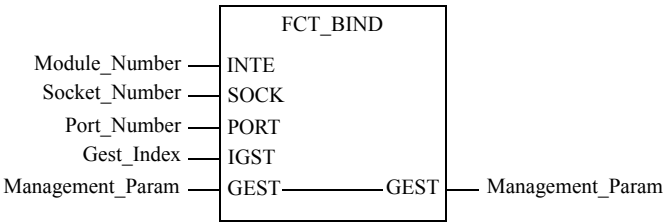
**Function Description**

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:

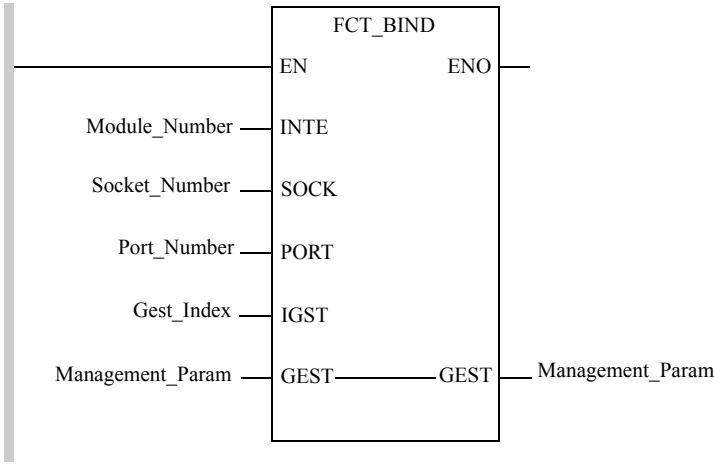
- 20 and 21 (FTP ports)
- 23 (Telnet port)
- 67 and 68 (BOOTP DHCP ports)
- 80 (HTTP port)
- 161 and 162 (SNMP ports)
- 502 (Schneider Electric port)
- 5000 and 5001 (specific ports of module)
- 1024 (TCP USER ports)
- 3124 (I/O port)
- 7400-8400 (RTPS ports)

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);
```

## Parameters

The following table describes the input parameters:

Parameter	Type	Comment
Module_Number	INT	Slot number of network module in rack 0. <ul style="list-style-type: none"> <li>● low byte is the slot number of network module in rack 0</li> <li>● high byte can be used to extend the number of sockets               <ul style="list-style-type: none"> <li>● 00 - provided for full backward compatibility with applications created on firmware version 3.3 or earlier</li> <li>● 01- up to 64 sockets can be used (firmware version must be higher than 3.3)</li> </ul> </li> </ul>
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 <i>p. 30</i> ) The operation report can have the following values: <ul style="list-style-type: none"> <li>● 16#00 : no error</li> <li>● 16#09 : the socket number is invalid</li> <li>● 16#16 : the socket is already bound</li> <li>● 16#30 : the specified port is already in use</li> <li>● 16#37 : the specified port number is not available</li> <li>● 16#41 : no route to host</li> </ul>



---

# 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.

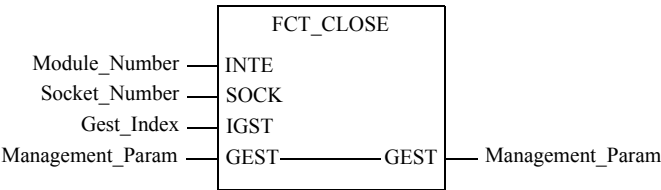
Because the sockets were opened with the SO\_LINGER option when using FCT\_SOCKET, the FCT\_CLOSE function is not blocked, even if the queues have not yet been sent or been acknowledged. This is called a hard or abortive close, because the socket's virtual circuit is reset immediately, and any unsent 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).

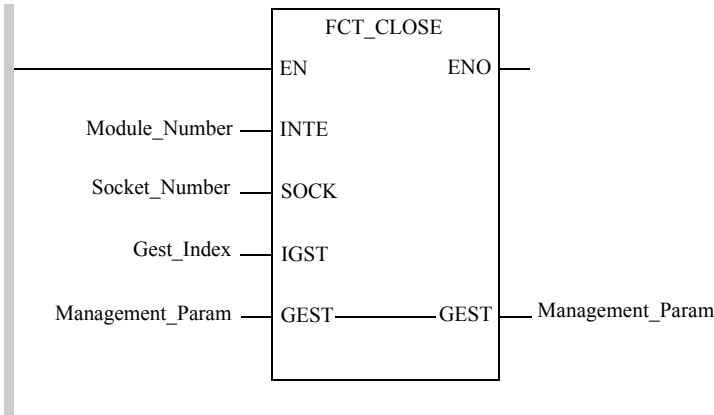
The additional parameters EN and ENO may be configured.

---

## 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);
```

**Parameters**

The following table describes the input parameters:

Parameter	Type	Comment
Module_Number	INT	Slot number of network module in rack 0. <ul style="list-style-type: none"><li>● low byte is the slot number of network module in rack 0</li><li>● high byte can be used to extend the number of sockets<ul style="list-style-type: none"><li>● 00 - provided for full backward compatibility with applications created on firmware version 3.3 or earlier</li><li>● 01- up to 64 sockets can be used (firmware version must be higher than 3.3)</li></ul></li></ul>
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 <i>p. 30</i> ) The operation report can have the following values: <ul style="list-style-type: none"><li>● 16#00: no error</li><li>● 16#16: the socket number is invalid</li></ul>



---

# FCT\_CONNECT: Establishes a connection with an IP address

10

---

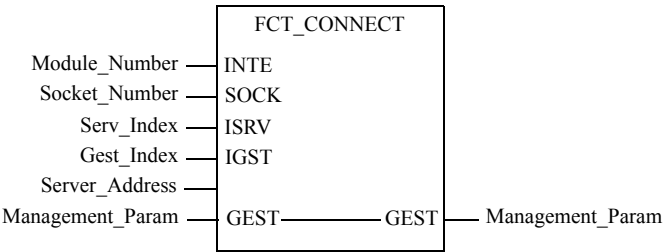
## Description

### Function Description

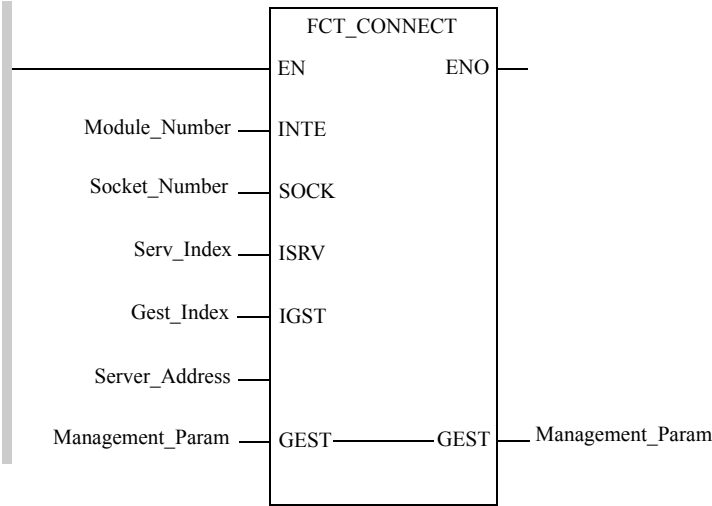
The `FCT_CONNECT` function is used to establish a connection to a known port and internet address.

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, Management\_Param

---

**Representation  
in ST**

FCT\_CONNECT(Module\_Number, Socket\_Number, Serv\_Index,  
Gest\_Index, Server\_Address, Management\_Param);

---

## Parameters

The following table describes the input parameters:

Parameter	Type	Comment
Module_Number	INT	Slot number of network module in rack 0. <ul style="list-style-type: none"> <li>● low byte is the slot number of network module in rack 0</li> <li>● high byte can be used to extend the number of sockets               <ul style="list-style-type: none"> <li>● 00 - provided for full backward compatibility with applications created on firmware version 3.3 or earlier</li> <li>● 01- up to 64 sockets can be used (firmware version must be higher than 3.3)</li> </ul> </li> </ul>
Socket_Number	INT	Socket number
Serv_Index	INT	Index of first word in <code>Server_Address</code> array
Gest_Index	INT	Index of first word in <code>Management_Param</code> 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 <i>p. 30</i> ) The operation report can have the following values: <ul style="list-style-type: none"> <li>● 16#00 : correct operation</li> <li>● 16#09 : invalid socket number</li> <li>● 16#16 : invalid parameter</li> <li>● 16#20 : connection is cut</li> <li>● 16#24 : connection in progress</li> <li>● 16#38 : socket already connected</li> <li>● 16#3D : connection refused</li> <li>● 16#41 : no route to host</li> </ul>



# FCT\_LISTEN: Configuration of a socket await connection

11

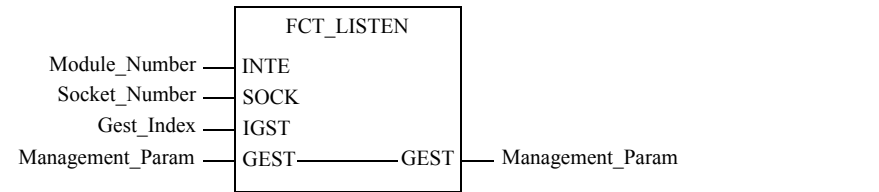
Description

Function Description

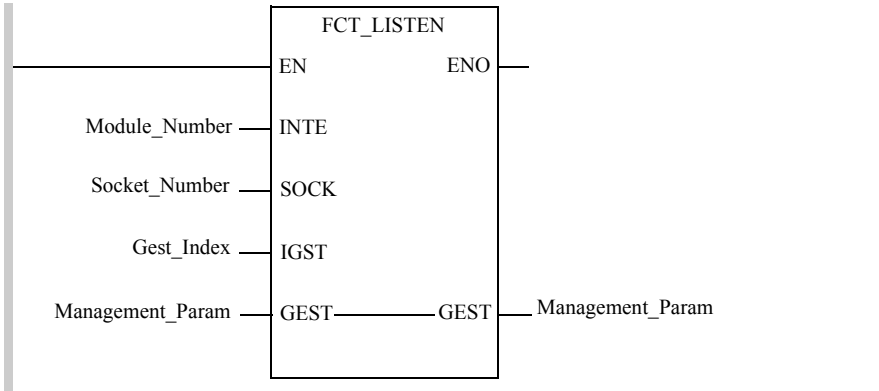
The `FCT_LISTEN` function sets up the specified socket to receive connections. Connection requests are queued on the socket until they are accepted with the `FCT_ACCEPT` call. The length of the queue is set to 16. If a connection request arrives while the queue is full, the requesting client gets an `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);

**Parameters** The following table describes the input parameters:

Parameter	Type	Comment
Module_Number	INT	Slot number of network module in rack 0. <ul style="list-style-type: none"><li>● low byte is the slot number of the network module in rack 0</li><li>● high byte can be used to extend the number of sockets<ul style="list-style-type: none"><li>● 00 - provided for full backward compatibility with applications created on firmware version 3.3 or earlier</li><li>● 01- up to 64 sockets can be used (firmware version must be higher than 3.3)</li></ul></li></ul>
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 <i>p. 30</i> ) The operation report can have the following values: <ul style="list-style-type: none"><li>● 16#00: no error</li><li>● 16#09: the socket number is invalid</li></ul>

# FCT\_RECEIVE: Retrieves data available on a socket

12

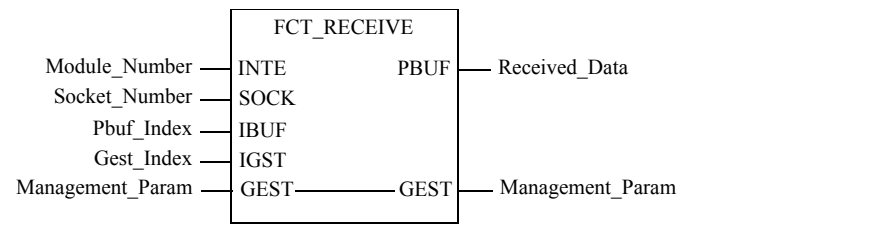
Description

**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.

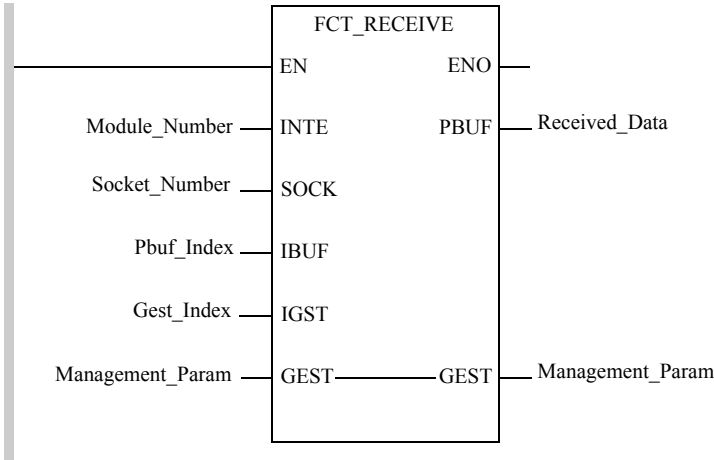
**Note:** FCT\_RECEIVE does not return out-of-band data.

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, Received\_Data

Representation  
in ST

FCT\_RECEIVE(Module\_Number, Socket\_Number, Pbuf\_Index,  
Gest\_Index, Management\_Param, Received\_Data);

Parameters

The following table describes the input parameters:

Parameter	Type	Comment
Module_Number	INT	Slot number of network module in rack 0. <ul style="list-style-type: none"><li>low byte is the slot number of the network module in rack 0</li><li>high byte can be used to extend the number of sockets<ul style="list-style-type: none"><li>00 - provided for full backward compatibility with applications created on firmware version 3.3 or earlier</li><li>01- up to 64 sockets can be used (firmware version must be higher than 3.3)</li></ul></li></ul>
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	<p>Function management array (see <i>p. 30</i>)</p> <p>The operation report can have the following values:</p> <ul style="list-style-type: none"><li>● 16#00 : no error</li><li>● 16#09 : the socket number is invalid</li><li>● 16#23 : no data to read</li><li>● 16#36 : the connection has been reset by peer</li><li>● 16#39 : the socket is not connected (listening socket)</li><li>● 16#3C: the keep alive timed out on broken connection</li><li>● 16#0E: the length of the character string to be received is greater than 240 bytes</li></ul> <p>The fourth word of the array should contain the number of bytes received if no error has occurred.</p>



---

## FCT\_SELECT: Multiplexes requests over sockets

13

---

### 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 two integers.

For ETY 5103 firmware revision 3.3 or lower:

- The socket descriptors are assigned a number from 1 to 32:
  - Numbers from 1 to 16 are assigned to sockets created by the socket function. They are listening sockets.
  - Numbers from 17 to 32 are assigned to sockets created by the accept function. They are connected sockets.
- The first word of the array corresponds to the listening sockets (bit 0 corresponds to socket 0) and the second word corresponds to the connected sockets.

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 as to listening versus connected sockets.

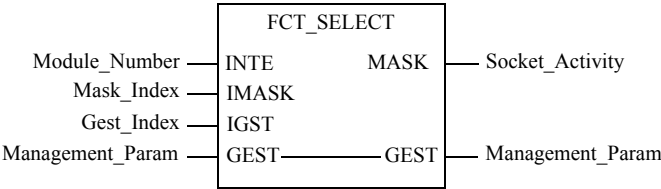
**Note:** For versions higher than 3.3, a length parameter is required in the `Management_Param` structure.

The length 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.

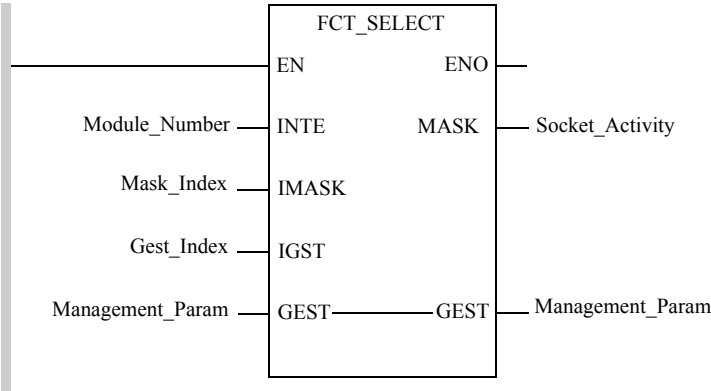
The additional parameters `EN` and `ENO` may be configured.

---

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);
```

## Parameters

The following table describes the input parameters:

Parameter	Type	Comment
Module_Number	INT	Slot number of network module in rack 0. <ul style="list-style-type: none"> <li>● low byte is the slot number of network module in rack 0</li> <li>● high byte can be used to extend the number of sockets               <ul style="list-style-type: none"> <li>● 00 - provided for full backward compatibility with applications created on firmware version 3.3 or earlier</li> <li>● 01- up to 64 sockets can be used (firmware version must be higher than 3.3)</li> </ul> </li> </ul>
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 p. 30) 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	Status of each socket. Each bit set to 1 indicates an event on the socket which corresponds to this bit. For example: <ul style="list-style-type: none"> <li>● if bit 3 of the second word has the value 1, socket 20 shall be read by the <code>FCT_RECEIVE</code> function</li> <li>● if bit 5 of the first word has the value 1, socket 6 shall be read by the <code>FCT_ACCEPT</code> function</li> </ul> <p>If a bit is set to 1 on a listening socket, this means that a connection request has been placed in the queue. If a bit has been set to 1 on an output socket, this means that data is queued on the socket or that communication has been interrupted.</p>



# FCT\_SEND: Sending data to a specified socket

14

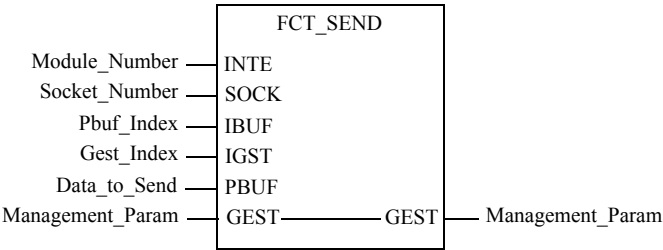
Description

**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.

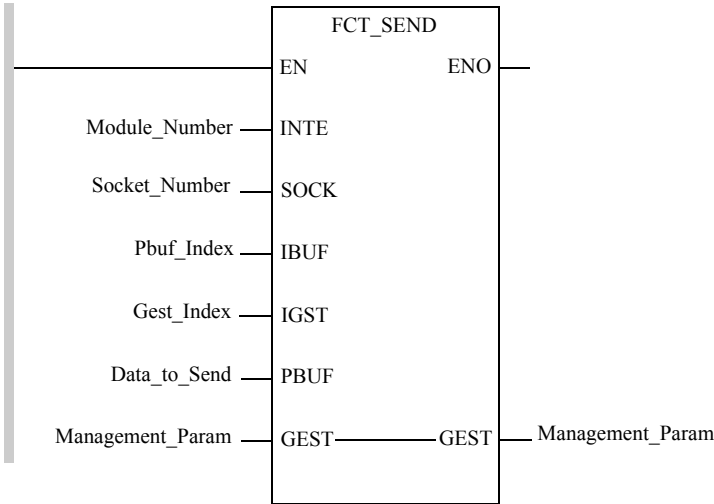
**Note:** it is not possible to send out of band data.

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);
```

## Parameters

The following table describes the input parameters:

Parameter	Type	Comment
Module_Number	INT	Slot number of network module in rack 0. <ul style="list-style-type: none"> <li>● Low byte is the slot number of network module in rack 0</li> <li>● High byte can be used to extend the number of sockets               <ul style="list-style-type: none"> <li>● 00 - provided for full backward compatibility with applications created on firmware version 3.3 or earlier</li> <li>● 01- up to 64 sockets can be used (firmware version must be higher than 3.3)</li> </ul> </li> </ul>
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 <i>p. 30</i> ) The operation report can have the following values: <ul style="list-style-type: none"> <li>● 16#00: no error</li> <li>● 16#09: the socket number is invalid</li> <li>● 16#23: the socket is full</li> <li>● 16#36: the connection has been reset by peer</li> <li>● 16#39: the socket is not connected (listening socket)</li> <li>● 16#0E: the length of the character string to be sent is greater than 240 bytes</li> </ul> The fourth word of the array should contain the number of bytes sent if no error has occurred.



---

# FCT\_SETSOCKOPT: Sets the options associated with the socket

15

---

## 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 p. 83) function.

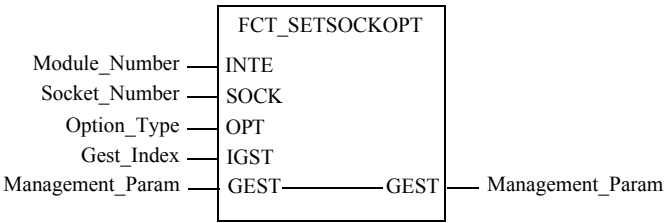
The following options are available:

- `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`.

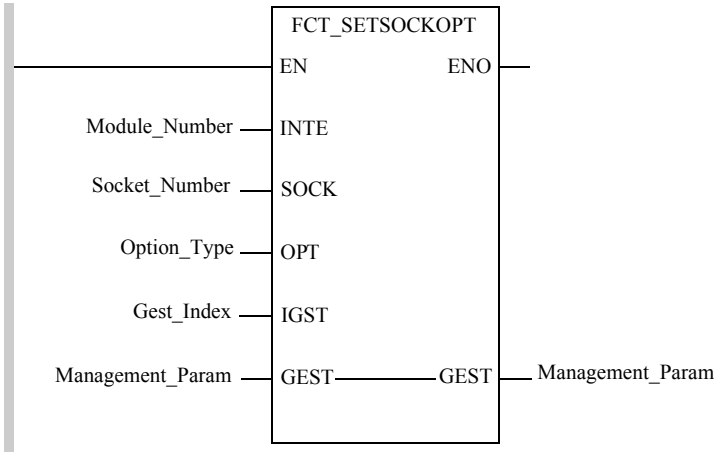
These options are selected by assigning a number in the `Option_Type` variable. The additional parameters `EN` and `ENO` may be configured.

---

## 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, Management\_Param);

**Parameters**

The following table describes the input parameters:

Parameter	Type	Comment
Module_Number	INT	Slot number of network module in rack 0. <ul style="list-style-type: none"> <li>● Low byte is the slot number of network module in rack 0</li> <li>● High byte can be used to extend the number of sockets <ul style="list-style-type: none"> <li>● 00 - provided for full backward compatibility with applications created on firmware version 3.3 or earlier</li> <li>● 01- up to 64 sockets can be used (firmware version must be higher than 3.3)</li> </ul> </li> </ul>
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: <ul style="list-style-type: none"> <li>● 1 for DONT_ROUTE</li> <li>● 2 for RESET_DONT_ROUTE</li> <li>● 3 for KEEP_ALIVE</li> <li>● 4 for RESET_KEEP_ALIVE</li> </ul>
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 <i>p. 30</i> ) The operation report can have the following values: <ul style="list-style-type: none"> <li>● 16#00 : no error</li> <li>● 16#09 : the socket number is invalid</li> <li>● 16#16 : invalid option</li> </ul> The fourth word of the array should contain the number of bytes stored in the buffer.



---

# FCT\_SHUTDOWN: Disables transmission on the socket

16

---

## Description

### Function Description

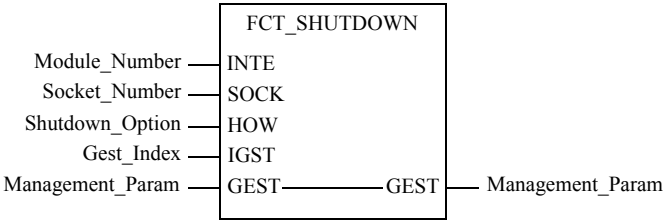
The `FCT_SHUTDOWN` function is used to disable either send/receive transmission on the socket. The TCP window is not changed and incoming data will be accepted (but not acknowledged) until the window is exhausted.

**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.

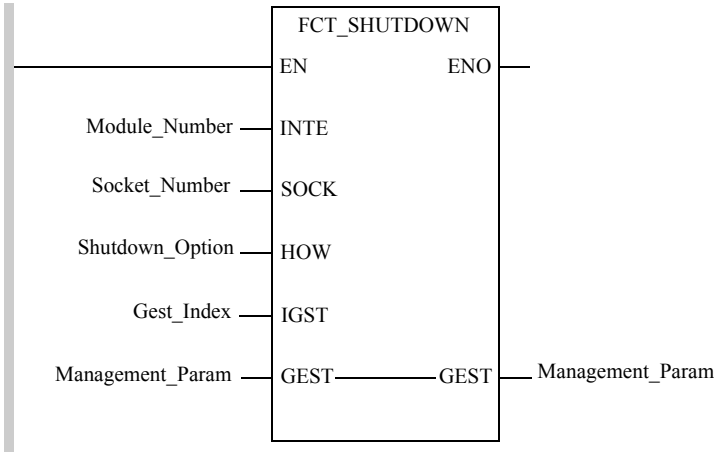
The additional parameters `EN` and `ENO` may be configured.

---

## 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);

**Parameters**

The following table describes the input parameters:

Parameter	Type	Comment
Module_Number	INT	Slot number of network module in rack 0. <ul style="list-style-type: none"> <li>● Low byte is the slot number of network module in rack 0</li> <li>● High byte can be used to extend the number of sockets               <ul style="list-style-type: none"> <li>● 00 - provided for full backward compatibility with applications created on firmware version 3.3 or earlier</li> <li>● 01- up to 64 sockets can be used (firmware version must be higher than 3.3)</li> </ul> </li> </ul>
Socket_Number	INT	Socket number
Shutdown_Option	INT	Disable transmission option: <ul style="list-style-type: none"> <li>● 0: no further receives are allowed on the socket</li> <li>● 1: no further sends are allowed on the socket, a FIN message is sent</li> <li>● 2: no further sends or receives are allowed on the socket. This option does the same as both of the previous options together.</li> </ul>
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 <i>p. 30</i> ) The operation report can have the following values: <ul style="list-style-type: none"> <li>● 16#00: no error</li> <li>● 16#09: the socket number is invalid</li> <li>● 16#16: an argument is not valid</li> <li>● 16#39: the socket is not connected</li> </ul>



---

# FCT\_SOCKET: Creation of a new socket

17

---

## Description

### Function Description

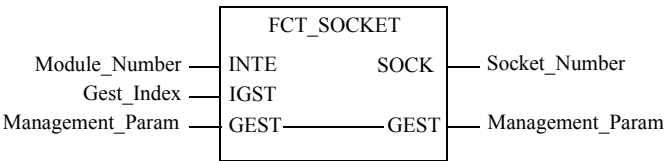
The `FCT_SOCKET` function creates a new socket and returns its socket number. The socket is a TCP/IP communication entity. It is created as a `STREAM TCP` socket with the following options:

- `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.

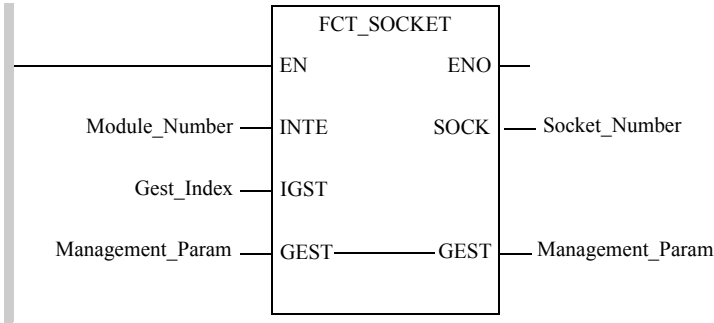
The additional parameters `EN` and `ENO` may be configured.

---

## 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);
```

**Parameters**

The following table describes the input parameters:

Parameter	Type	Comment
Module_Number	INT	Slot number of network module in rack 0. <ul style="list-style-type: none"><li>• Low byte is the slot number of network module in rack 0</li><li>• High byte can be used to extend the number of sockets<ul style="list-style-type: none"><li>• 00 - provided for full backward compatibility with applications created on firmware version 3.3 or earlier</li><li>• 01- up to 64 sockets can be assigned (firmware version must be higher than 3.3)</li></ul></li></ul>
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 <i>p. 30</i> ) The operation report can have the following values: <ul style="list-style-type: none"><li>• 16#00: no error</li><li>• 16#37: the maximum number of sockets has been reached</li></ul>



---

# Appendices



---

## Introduction

### What's in this Appendix?

The appendix contains the following chapters:

Chapter	Chapter Name	Page
A	System objects	89

---



---

# System objects



---

## At a Glance

### Subject of this Chapter

This appendix describes the system bits and words of Unity Pro language.

**Note:** The symbols, associated with each bit object or system word, mentioned in the descriptive tables of these objects, are not implemented as standard in the software, but can be entered using the data editor.

They are proposed in order to ensure the homogeneity of their names in the different applications.

### What's in this Chapter?

This appendix contains the following topics:

Topic	Page
System bit introduction	90
Description of system bits %S15 to %S21	91
Description of System Words %SW12 to %SW19	94

## System bit introduction

---

### General

The Premium, Atrium and Quantum PLCs use %Si system bits which indicate the state of the PLC, or they can be used to control how it operates.

These bits can be tested in the user program to detect any functional development requiring a set processing procedure.

Some of these bits must be reset to their initial or normal state by the program. However, the system bits that are reset to their initial or normal state by the system must not be reset by the program or by the terminal.

---

## Description of system bits %S15 to %S21

### System Bits %S15 to %S21

Bit Symbol	Function	Description	Initial state	Quantum	Premium Atrium
<b>%S15</b> STRINGERROR	Character string fault	Normally set to 0, this bit is set to 1 when the destination zone for a character string transfer is not of sufficient size (including the number of characters and the end of string character) to receive this character string. The application stops in error state if the %S78 bit has been to set to 1. This bit must be reset to 0 by the application.	0	YES	YES
<b>%S16</b> IOERRTSK	Task input/ output fault	Normally set to 1, this is set to 0 by the system when a fault occurs on an in-rack I/O module or a Fipio device configured in the task. This bit must be reset to 1 by the user.	1	YES	YES
<b>%S17</b> CARRY	Rotate shift output	Normally at 0. During a rotate shift operation, this takes the state of the outgoing bit.	0	YES	YES

Bit Symbol	Function	Description	Initial state	Quantum	Premium Atrium
<b>%S18</b> OVERFLOW	Overflow or arithmetic error	<p>Normally set to 0, this is set to 1 in the event of a capacity overflow if there is:</p> <ul style="list-style-type: none"> <li>• a result greater than + 32 767 or less than - 32 768, in single length,</li> <li>• result greater than + 65 535, in unsigned integer,</li> <li>• a result greater than + 2 147 483 647 or less than - 2 147 483 648, in double length,</li> <li>• result greater than +4 294 967 296, in double length or unsigned integer,</li> <li>• real values outside limits,</li> <li>• division by 0,</li> <li>• the root of a negative number,</li> <li>• forcing to a non-existent step on a drum.</li> <li>• stacking up of an already full register, emptying of an already empty register.</li> </ul> <p>It must be tested by the user program after each operation where there is a risk of overflow, then reset to 0 by the user if there is indeed an overflow. When the %S18 bit switches to 1, the application stops in error state if the %S78 bit has been to set to 1.</p>	0	YES	YES
<b>%S19</b> OVERRUN	Task period overrun (periodical scanning)	<p>Normally set to 0, this bit is set to 1 by the system in the event of a time period overrun (i.e. task execution time is greater than the period defined by the user in the configuration or programmed into the %SW word associated with the task). The user must reset this bit to 0. Each task manages its own %S19 bit.</p>	0	YES	YES
<b>%S20</b> INDEXOVF	Index overflow	<p>Normally set to 0, this is set to 1 when the address of the indexed object becomes less than 0 or exceeds the number of objects declared in the configuration.</p> <p>In this case, it is as if the index were equal to 0.</p> <p>It must be tested by the user program after each operation where there is a risk of overflow, then reset to 0 if there is indeed an overflow.</p> <p>When the %S20 bit switches to 1, the application stops in error state if the %S78 bit has been to set to 1.</p>	0	YES	YES

Bit Symbol	Function	Description	Initial state	Quantum	Premium Atrium
<b>%S21</b> 1RSTTASKRUN	First task cycle	Tested in a task (Mast, Fast, Aux0, Aux1, Aux2 Aux3), the bit %S21 indicates the first cycle of this task. %S21 is set to 1 at the start of the cycle and reset to zero at the end of the cycle. <b>Notes:</b> the bit %S21 does not have the same meaning in PL7 as in Unity Pro.	0	YES	YES

## CAUTION

### **%S16 for Quantum PLCs**

On Quantum, communication errors from modules (NOM, NOE, NWM, CRA, CRP) and MMS modules are not reported on bits %S10 and %S16.

It is entirely your responsibility to ensure that these system bits are used correctly

**Failure to follow this instruction can result in injury or equipment damage.**

## Description of System Words %SW12 to %SW19

### System Words %SW12 to %SW19

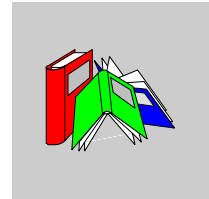
Word Symbol	Function	Description	Initial state	Quantum	Premium Atrium
<b>%SW12</b> UTWPORTADDR	Uni-Telway terminal port address	Uni-Telway address of terminal port (in slave mode) as defined in the configuration and loaded into this word on cold start. <b>Note:</b> The modification of the value of this word is not taken into account by the system	-	NO	YES
<b>%SW13</b> XWAYNETWADDR	Main address of the station	Indicates the following for the main network (Fipway or Ethway): <ul style="list-style-type: none"> <li>the station number (least significant byte) from 0 to 127,</li> <li>the network number (most significant byte) from 0 to 63,</li> </ul> (value of the micro-switches on the PCMCIA card).	254 (16#00FE)	NO	YES
<b>%SW14</b> OSCOMMVERS	Commerci al version of PLC processor	This word contains the commercial version of the PLC processor. <b>Example:</b> 16#0135 version: 01 issue number: 35	-	YES	YES
<b>%SW15</b> OSCOMPATCH	PLC processor patch version	This word contains the commercial version of the PLC processor patch. It is coded onto the least significant byte of the word. Coding: 0 = no patch, 1 = A, 2 = B... <b>Example:</b> 16#0003 corresponds to patch C.	-	YES	YES
<b>%SW16</b> OSINTVERS	Firmware version number	This word contains the Firmware version number in hexadecimal of the PLC processor firmware. <b>Example:</b> 16#0017 version: 2.1 VN: 17	-	YES	YES

Word Symbol	Function	Description	Initial state	Quantum	Premium Atrium
<b>%SW17</b> FLOATSTAT	Errorstatus on floating operation	<p>On detection of an error in a floating arithmetic operation, bit %S18 is set to 1 and %SW17 error status is updated according to the following coding:</p> <ul style="list-style-type: none"> <li>● %SW17.0 = Invalid operation / result is not a number</li> <li>● %SW17.1 = Non-standardized operand / result is acceptable</li> <li>● %SW17.2 = Division by 0 / result is infinity</li> <li>● %SW17.3 = Overflow / result is infinity</li> <li>● %SW17.4 = Underflow / result is 0</li> <li>● %SW17.5 to 15 = not used</li> </ul> <p>This word is reset to 0 by the system on cold start, and also by the program for re-usage purposes.</p>	0	YES	YES
<b>%SW18</b> <b>%SW19</b> 100MSCOUNTER	Absolute time counter	<p>The words %SW18 and %SW19 are used to calculate duration.</p> <p>They are incremented every 1/10<sup>th</sup> of a second by the system (even when PLC is in STOP, they are no longer incremented if the PLC is powered down). They can be read and written by the user program or by the terminal.</p>	0	YES	YES



---

# Glossary



---

## !

<b>%I</b>	According to the IEC standard, %I indicates a discrete input-type language object.
<b>%IW</b>	According to the IEC standard, %IW indicates an analog input -type language object.
<b>%KW</b>	According to the IEC standard, %KW indicates a constant word-type language object.
<b>%M</b>	According to the IEC standard, %M indicates a memory bit-type language object.
<b>%MW</b>	According to the IEC standard, %MW indicates a memory word-type language object.
<b>%Q</b>	According to the IEC standard, %Q indicates a discrete output-type language object.
<b>%QW</b>	According to the IEC standard, %QW indicates an analog output-type language object.

---

## A

<b>ADDR_TYPE</b>	This predefined type is used as output for ADDR function. This type is ARRAY[0..5] OF Int. You can find it in the libset, in the same family than the EFs which use it.
<b>ANL_IN</b>	ANL_IN is the abbreviation of Analog Input data type and is used when processing analog values. The %IW addresses for the configured analog input module, which were specified in the I/O component list, are automatically assigned data types and should therefore only be occupied with Unlocated Variables.

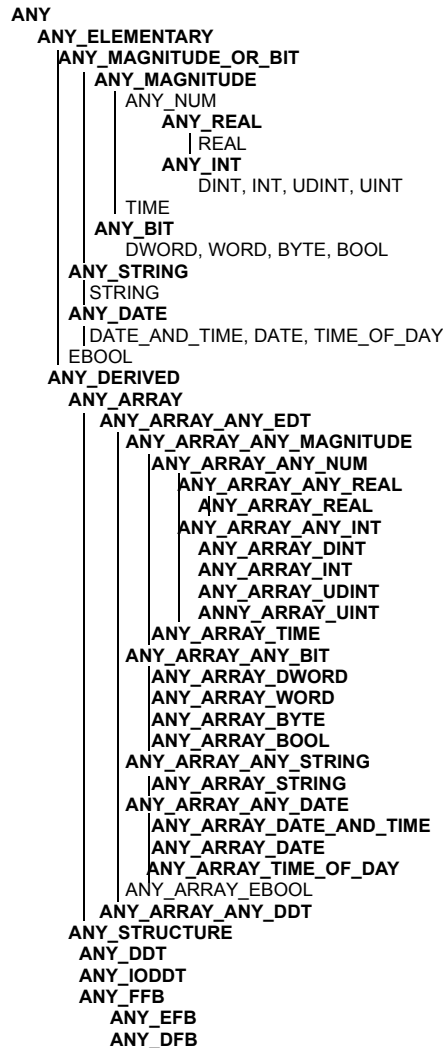
**ANL\_OUT**

ANL\_OUT is the abbreviation of Analog Output data type and is used when processing analog values. The %MW addresses for the configured analog input module, which were specified in the I/O component list, are automatically assigned data types and should therefore only be occupied with Unlocated Variables.

**ANY**

There is a hierarchy between the different types of data. In the DFB, it is sometimes possible to declare which variables can contain several types of values. Here, we use ANY\_xxx types.

The following diagram shows the hierarchically-ordered structure:



**ARRAY**

An **ARRAY** is a table of elements of the same type.

The syntax is as follows: `ARRAY [<terminals>] OF <Type>`

Example:

`ARRAY [1..2] OF BOOL` is a one-dimensional table made up of two **BOOL**-type elements.

`ARRAY [1..10, 1..20] OF INT` is a two-dimensional table made up of 10x20 **INT**-type elements.

**B****Base 10 literals**

A literal value in base 10 is used to represent a decimal integer value. This value can be preceded by the signs "+" and "-". If the character "\_" is employed in this literal value, it is not significant.

Example:

`-12, 0, 123_456, +986`

**Base 16 Literals**

A literal value in base 16 is used to represent an integer in hexadecimal. The base is determined by the number "16" and the sign "#". The signs "+" and "-" are not allowed. For greater clarity when reading, you can use the sign "\_" between bits.

Example:

`16#F_F` or `16#FF` (in decimal 255)

`16#F_F` or `16#FF` (in decimal 224)

**Base 2 Literals**

A literal value in base 2 is used to represent a binary integer. The base is determined by the number "2" and the sign "#". The signs "+" and "-" are not allowed. For greater clarity when reading, you can use the sign "\_" between bits.

Example:

`2#1111_1111` or `2#11111111` (in decimal 255)

`2#1110_0000` or `2#11100000` (in decimal 224)

**Base 8 Literals**

A literal value in base 8 is used to represent an octal integer. The base is determined by the number "8" and the sign "#". The signs "+" and "-" are not allowed. For greater clarity when reading, you can use the sign "\_" between bits.

Example:

`8#3_77` or `8#377` (in decimal 255)

`8#34_0` or `8#340` (in decimal 224)

**BCD**

**BCD** is the abbreviation of Binary Coded Decimal format

**BCD** is used to represent decimal numbers between 0 and 9 using a group of four bits (half-byte).

In this format, the four bits used to code the decimal numbers have a range of unused combinations.

Example of BCD coding:

- the number 2450
- is coded: 0010 0100 0101 0000

**BOOL** **BOOL** is the abbreviation of Boolean type. This is the elementary data item in computing. A **BOOL** type variable has a value of either: 0 (**FALSE**) or 1 (**TRUE**). A **BOOL** type word extract bit, for example: %MW10.4.

**BYTE** When 8 bits are put together, this is called a **BYTE**. A **BYTE** is either entered in binary, or in base 8.  
The **BYTE** type is coded in an 8 bit format, which, in hexadecimal, ranges from 16#00 to 16#FF

---

**D**

**DATE** The **DATE** type coded in BCD in 32 bit format contains the following information:

- the year coded in a 16-bit field,
- the month coded in an 8-bit field,
- the day coded in an 8-bit field.

The **DATE** type is entered as follows: **D#<Year>-<Month>-<Day>**

This table shows the lower/upper limits in each field:

Field	Limits	Comment
Year	[1990,2099]	Year
Month	[01,12]	The left 0 is always displayed, but can be omitted at the time of entry
Day	[01,31]	For the months 01\03\05\07\08\10\12
	[01,30]	For the months 04\06\09\11
	[01,29]	For the month 02 (leap years)
	[01,28]	For the month 02 (non leap years)

**DATE\_AND\_TIME** see **DT**

**DBCD** Representation of a Double BCD-format double integer.  
The Binary Coded Decimal (BCD) format is used to represent decimal numbers between 0 and 9 using a group of four bits.  
In this format, the four bits used to code the decimal numbers have a range of unused combinations.

Example of DBCD coding:

- the number 78993016
- is coded: 0111 1000 1001 1001 0011 0000 0001 0110

**DDT**

DDT is the abbreviation of Derived Data Type.

A derived data type is a set of elements of the same type (ARRAY) or of various types (structure)

**DFB**

DFB is the abbreviation of Derived Function Block.

DFB types are function blocks that can be programmed by the user ST, IL, LD or FBD.

By using DFB types in an application, it is possible to:

- simplify the design and input of the program,
- increase the legibility of the program,
- facilitate the debugging of the program,
- reduce the volume of the generated code.

**DINT**

DINT is the abbreviation of Double Integer format (coded on 32 bits).

The lower and upper limits are as follows:  $-(2 \text{ to the power of } 31)$  to  $(2 \text{ to the power of } 31) - 1$ .

Example:

-2147483648, 2147483647, 16#FFFFFFFF.

**DT**

DT is the abbreviation of Date and Time.

The DT type coded in BCD in 64 bit format contains the following information:

- The year coded in a 16-bit field,
- the month coded in an 8-bit field,
- the day coded in an 8-bit field,
- the hour coded in a 8-bit field,
- the minutes coded in an 8-bit field,
- the seconds coded in an 8-bit field.

**Note:** The 8 least significant bits are unused.

The DT type is entered as follows:

**DT#**<Year>-<Month>-<Day>-<Hour>:<Minutes>:<Seconds>

This table shows the lower/upper limits in each field:

Field	Limits	Comment
Year	[1990,2099]	Year
Month	[01,12]	The left 0 is always displayed, but can be omitted at the time of entry

Field	Limits	Comment
Day	[01,31]	For the months 01\03\05\07\08\10\12
	[01,30]	For the months 04\06\09\11
	[01,29]	For the month 02 (leap years)
	[01,28]	For the month 02 (non leap years)
Hour	[00,23]	The left 0 is always displayed, but can be omitted at the time of entry
Minute	[00,59]	The left 0 is always displayed, but can be omitted at the time of entry
Second	[00,59]	The left 0 is always displayed, but can be omitted at the time of entry

## DWORD

**DWORD** is the abbreviation of Double Word.

The **DWORD** type is coded in 32 bit format.

This table shows the lower/upper limits of the bases which can be used:

Base	Lower limit	Upper limit
Hexadecimal	16#0	16#FFFFFFFF
Octal	8#0	8#3777777777
Binary	2#0	2#11111111111111111111111111111111

Representation examples:

Data content	Representation in one of the bases
00000000000010101101110011011110	16#ADCDE
000000000000001000000000000000	8#200000
00000000000010101011110011011110	2#10101011110011011110

---

## E

## EBOOL

**EBOOL** is the abbreviation of Extended Boolean type. A **EBOOL** type variable brings a value (0 (FALSE) or 1 (TRUE)) but also rising or falling edges and forcing capabilities.

An **EBOOL** type variable takes up one byte of memory.

The byte split up into:

- one bit for the value,

- one bit for the history bit (each time the state's object changes, the value is copied inside the history bit),
- one bit for the forcing bit (equals to 0 if the object isn't forced, equal to 1 if the bit is forced).

The default type value of each bit is 0 (**FALSE**).

**EF**

Is the abbreviation of Elementary Function.

This is a block which is used in a program, and which performs a predefined software function.

A function has no internal status information. Multiple invocations of the same function using the same input parameters always supply the same output values. Details of the graphic form of the function invocation can be found in the "[Functional block (instance)] ". In contrast to the invocation of the function blocks, function invocations only have a single unnamed output, whose name is the same as the function. In FBD each invocation is denoted by a unique [number] via the graphic block, this number is automatically generated and can not be altered.

You position and set up these functions in your program in order to carry out your application.

You can also develop other functions using the SDKC development kit.

**EFB**

Is the abbreviation for Elementary Function Block.

This is a block which is used in a program, and which performs a predefined software function.

EFBs have internal statuses and parameters. Even where the inputs are identical, the output values may be different. For example, a counter has an output which indicates that the preselection value has been reached. This output is set to 1 when the current value is equal to the preselection value.

**Elementary Function**

see EF

**EN**

**EN** means **EN**able, this is an optional block input. When **EN** is activated, an **ENO** output is automatically drafted.

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

If **EN** = 1, the internal program of the block is executed, and **ENO** is set to 1 by the system. If an error occurs, **ENO** is set to 0.

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

**ENO**

**ENO** means **Error NOT**ification, this is the output associated to the optional input **EN**. If **ENO** is set to 0 (caused by **EN**=0 or in case of an execution error),

- the outputs of function blocks remain in the status they were in for the last correct executed scanning cycle and

- the output(s) of functions and procedures are set to "0".
- 

**F****FBD**

FBD is the abbreviation of Function Block Diagram.

FBD is a graphic programming language that operates as a logic diagram. In addition to the simple logic blocks (AND, OR, etc.), each function or function block of the program is represented using this graphic form. For each block, the inputs are located to the left and the outputs to the right. The outputs of the blocks can be linked to the inputs of other blocks to form complex expressions.

**FFB**

Collective term for EF (Elementary Function), EFB (Elementary Function Block) and DFB (Derived Function block)

**Function**

see EF

**Function Block Diagram**

see FBD

---

**G****GRAY**

Gray or "reflected binary" code is used to code a numerical value being developed into a chain of binary configurations that can be differentiated by the change in status of one and only one bit.

This code can be used, for example, to avoid the following random event: in pure binary, the change of the value 0111 to 1000 can produce random numbers between 0 and 1000, as the bits do not change value altogether simultaneously.

Equivalence between decimal, BCD and Gray:

Decimal	0	1	2	3	4	5	6	7	8	9
BCD	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001
Gray	0000	0001	0011	0010	0110	0111	0101	0100	1100	1101

---

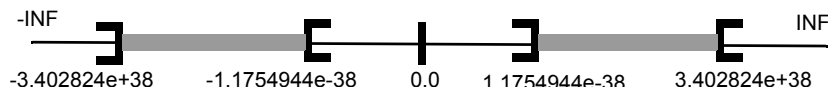
---

**I**

**IEC 61131-3** International standard: Programmable Logic Controls  
Part 3: Programming languages.

**IL** IL is the abbreviation of Instruction List.  
This language is a series of basic instructions.  
This language is very close to the assembly language used to program processors.  
Each instruction is composed of an instruction code and an operand.

**INF** Used to indicate that a number overruns the allowed limits.  
For a number of Integers, the value ranges (shown in gray) are as follows:



When a calculation result is:

- less than  $-3.402824e+38$ , the symbol  $-INF$  (for -infinite) is displayed,
- greater than  $+3.402824e+38$ , the symbol  $INF$  (for +infinite) is displayed.

**INT** **INT** is the abbreviation of single integer format (coded on 16 bits).  
The lower and upper limits are as follows:  $-(2 \text{ to the power of } 15) \text{ to } (2 \text{ to the power of } 15) - 1$ .  
Example:  
 $-32768, 32767, 2\#1111110001001001, 16\#9FA4$ .

**Integer Literals** Integer literal are used to enter integer values in the decimal system. The values can have a preceding sign (+/-). Individual underlines ( \_ ) between numbers are not significant.  
Example:  
 $-12, 0, 123\_456, +986$

**IODDT** IODDT is the abbreviation of Input/Output Derived Data Type.  
The term IODDT designates a structured data type representing a module or a channel of a PLC module. Each application expert module possesses its own IODDTs.

---

**K**

**Keyword** A keyword is a unique combination of characters used as a syntactical programming language element (See annex B definition of the IEC standard 61131-3. All the key words used in Unity Pro and of this standard are listed in annex C of the IEC standard 61131-3. These keywords cannot be used as identifiers in your program (names of variables, sections, DFB types, etc.)).

---

**L**

**LD** LD is the abbreviation of Ladder Diagram.  
LD is a programming language, representing the instructions to be carried out in the form of graphic diagrams very close to a schematic electrical diagram (contacts, coils, etc.).

**Located variables** A located variable is a variable for which it is possible to know its position in the PLC memory. For example, the variable `Water_pressure`, is associated with `%MW102`. `Water_pressure` is said to be localized.

---

**M**

**Multiple Token** Operating mode of an SFC. In multitoken mode, the SFC may possess several active steps at the same time.

---

**N**

**Naming conventions (Identifier)** An identifier is a sequence of letters, numbers and underlines beginning with a letter or underline (e.g. name of a function block type, an instance, a variable or a section). Letters from national character sets (e.g: ö, ü, é, ò) can be used except in project and DFB names. Underlines are significant in identifiers; e.g. `A_BCD` and `AB_CD` are interpreted as different identifiers. Multiple leading underlines and consecutive underlines are invalid.  
Identifiers cannot contain spaces. Not case sensitive; e.g. `ABCD` and `abcd` are interpreted as the same identifier.

According to IEC 61131-3 leading digits are not allowed in identifiers. Nevertheless, you can use them if you activate in dialog **Tools** → **Project settings** in tab **Language extensions** the check box **Leading digits**.  
Identifiers cannot be keywords.

## NAN

Used to indicate that a result of an operation is not a number (NAN = Not A Number).  
Example: calculating the square root of a negative number.

**Note:** The IEC 559 standard defines two classes of NAN: quiet `NAN` (`QNaN`) and signaling `NaN` (`SNaN`). `QNaN` is a `NAN` with the most significant fraction bit set and a `SNaN` is a `NAN` with the most significant fraction bit clear (Bit number 22). `QNaNs` are allowed to propagate through most arithmetic operations without signaling an exception. `SNaN` generally signal an invalid-operation exception whenever they appear as operands in arithmetic operations (See %SW17 and %S18).

## Network

There are two meanings for Network.

- In LD:  
A network is a set of interconnected graphic elements. The scope of a network is local to the program organization unit (section) in which the network is located.
- With communication expert modules:  
A network is a group of stations which communicate among one another. The term network is also used to define a group of interconnected graphic elements. This group forms then a part of a program which may be composed of a group of networks.

## P

## Procedure

Procedures are functions view technically. The only difference to elementary functions is that procedures can take up more than one output and they support data type `VAR_IN_OUT`. To the eye, procedures are no different than elementary functions.

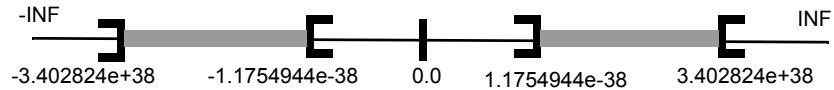
Procedures are a supplement to IEC 61131-3.

## R

## REAL

Real type is a coded type in 32 bits.

The ranges of possible values are illustrated in gray in the following diagram:



When a calculation result is:

- between  $-1.175494\text{e-}38$  and  $1.175494\text{e-}38$  it is considered as a `DEN`,
- less than  $-3.402824\text{e+}38$ , the symbol `-INF` (for - infinite) is displayed,
- greater than  $+3.402824\text{e+}38$ , the symbol `INF` (for +infinite) is displayed,
- undefined (square root of a negative number), the symbol `NAN` or `NAN` is displayed.

**Note:** The IEC 559 standard defines two classes of NAN: quiet NAN (`QNAN`) and signaling `NaN` (`SNAN`) `QNAN` is a NAN with the most significant fraction bit set and a `SNAN` is a NAN with the most significant fraction bit clear (Bit number 22). `QNANs` are allowed to propagate through most arithmetic operations without signaling an exception. `SNAN` generally signal an invalid-operation exception whenever they appear as operands in arithmetic operations (See %SW17 and %S18).

**Note:** when an operand is a `DEN` (Denormalized number) the result is not significant.

## Real Literals

An literal real value is a number expressed in one or more decimals.

Example:

`-12.0`, `0.0`, `+0.456`, `3.14159_26`

## Real Literals with Exponent

An Literal decimal value can be expressed using standard scientific notation. The representation is as follows: mantissa + exponential.

Example:

`-1.34E-12` or `-1.34e-12`  
`1.0E+6` or `1.0e+6`  
`1.234E6` or `1.234e6`

---

## S

## SFC

SFC is the abbreviation of Sequential Function Chart.

SFC enables the operation of a sequential automation device to be represented graphically and in a structured manner. This graphic description of the sequential behavior of an automation device, and the various situations which result from it, is performed using simple graphic symbols.

<b>Single Token</b>	Operating mode of an SFC chart for which only a single step can be active at any one time.
<b>ST</b>	ST is the abbreviation of Structured Text language. Structured Text language is an elaborated language close to computer programming languages. It enables you to structure series of instructions.
<b>STRING</b>	A variable of the type <code>STRING</code> is an ASCII standard character string. A character string has a maximum length of 65534 characters.

## T

<b>TIME</b>	The type <code>TIME</code> expresses a duration in milliseconds. Coded in 32 bits, this type makes it possible to obtain periods from 0 to $2^{32}-1$ milliseconds. The units of type <code>TIME</code> are the following: the days (d), the hours (h), the minutes (m), the seconds (s) and the milliseconds (ms). A literal value of the type <code>TIME</code> is represented by a combination of previous types preceded by <code>T#</code> , <code>t#</code> , <code>TIME#</code> or <code>time#</code> . Examples: <code>T#25h15m</code> , <code>t#14.7S</code> , <code>TIME#5d10h23m45s3ms</code>
<b>Time literals</b>	The units of type <code>TIME</code> are the following: the days (d), the hours (h), the minutes (m), the seconds (s) and the milliseconds (ms). A literal value of the type <code>TIME</code> is represented by a combination of previous types preceded by <code>T#</code> , <code>t#</code> , <code>TIME#</code> or <code>time#</code> . Examples: <code>T#25h15m</code> , <code>t#14.7S</code> , <code>TIME#5d10h23m45s3ms</code>
<b>TIME_OF_DAY</b>	see <code>TOD</code>
<b>TOD</b>	<code>TOD</code> is the abbreviation of Time of Day. The <code>TOD</code> type coded in BCD in 32 bit format contains the following information: <ul style="list-style-type: none"> <li>• the hour coded in a 8-bit field,</li> <li>• the minutes coded in an 8-bit field,</li> <li>• the seconds coded in an 8-bit field.</li> </ul>

**Note:** The 8 least significant bits are unused.

The Time of Day type is entered as follows: **TOD#**<Hour>:<Minutes>:<Seconds>  
This table shows the lower/upper limits in each field:

Field	Limits	Comment
Hour	[00,23]	The left 0 is always displayed, but can be omitted at the time of entry
Minute	[00,59]	The left 0 is always displayed, but can be omitted at the time of entry
Second	[00,59]	The left 0 is always displayed, but can be omitted at the time of entry

Example: TOD#23:59:45.

**Token** An active step of an SFC is known as a token.

**TOPO\_ADDR\_TY  
PE** This predefined type is used as output for READ\_TOPO\_ADDR function. This type is an ARRAY[0..4] OF Int. You can find it in the libset, in the same family than the EFs which use it.

---

## U

**UDINT** UDINT is the abbreviation of Unsigned Double Integer format (coded on 32 bits) unsigned. The lower and upper limits are as follows: 0 to (2 to the power of 32) - 1.  
Example:  
0, 4294967295, 2#11111111111111111111111111111111, 8#377777777777, 16#FFFFFFFF.

**UINT** UINT is the abbreviation of Unsigned integer format (coded on 16 bits). The lower and upper limits are as follows: 0 to (2 to the power of 16) - 1.  
Example:  
0, 65535, 2#1111111111111111, 8#177777, 16#FFFF.

**Unlocated  
variable** An unlocated variable is a variable for which it is impossible to know its position in the PLC memory. A variable which have no address assigned is said to be unlocated.

---

**V**

**Variable** Memory entity of the type `BOOL`, `WORD`, `DWORD`, etc., whose contents can be modified by the program during execution.

---

**W**

**WORD** The `WORD` type is coded in 16 bit format and is used to carry out processing on bit strings.  
This table shows the lower/upper limits of the bases which can be used:

Base	Lower limit	Upper limit
Hexadecimal	16#0	16#FFFF
Octal	8#0	8#177777
Binary	2#0	2#1111111111111111

Representation examples

Data content	Representation in one of the bases
0000000011010011	16#D3
1010101010101010	8#125252
0000000011010011	2#11010011

---



---

## Index

---



### Symbols

%S15, 91  
%S16, 91  
%S17, 91  
%S18, 92  
%S19, 92  
%S20, 92  
%S21, 93  
%SW12, 94  
%SW13, 94  
%SW14, 94  
%SW15, 94  
%SW16, 94  
%SW17, 95  
%SW18, 95  
%SW19, 95

### Numerics

100MSCOUNTER, 95  
1RSTTASKRUN, 93

### A

Accepts a connection request  
    FCT\_ACCEPT, 45

### Advanced

FCT\_ACCEPT, 45  
FCT\_BIND, 49  
FCT\_CLOSE, 53  
FCT\_CONNECT, 57  
FCT\_LISTEN, 61  
FCT\_RECEIVE, 63  
FCT\_SELECT, 67  
FCT\_SEND, 71  
FCT\_SETSOCKOPT, 75  
FCT\_SHUTDOWN, 79  
FCT\_SOCKET, 83

### B

Block types, 10

### C

CARRY, 91  
Client/server example  
    Open TCP, 35  
Client/server model  
    Open TCP, 33  
Conditional FFB Call, 15  
Configuration of a socket to await connection  
    FCT\_LISTEN, 61  
Creation of a socket  
    FCT\_SOCKET, 83

## D

- Debugging
  - TCP Open, 42
- Deletes the socket
  - FCT\_CLOSE, 53
- Derived function block, 10
- Diagnostics
  - TCP Open, 42
- Disables transmission on the socket
  - FCT\_SHUTDOWN, 79

## E

- Elementary Function, 10
- Elementary function block, 10
- EN, 14
- ENO, 14
- Establishes IP connection
  - FCT\_CONNECT, 57

## F

- FCT\_ACCEPT, 45
- FCT\_BIND, 49
- FCT\_CLOSE, 53
- FCT\_CONNECT, 57
- FCT\_LISTEN, 61
- FCT\_RECEIVE, 63
- FCT\_SELECT, 67
- FCT\_SEND, 71
- FCT\_SETSOCKOPT, 75
- FCT\_SHUTDOWN, 79
- FCT\_SOCKET, 83
- FLOATSTAT, 95

## I

- INDEXOVF, 92
- IOERRTSK, 91

## M

- Management parameters
  - Open TCP, 30

- Management table
  - Open TCP, 30
- Multiplexes requests over sockets
  - FCT\_SELECT, 67

## O

- Open TCP
  - Client/server example, 35
  - Client/server model, 33
  - Management table, 30
  - Report, 30
- Operating modes of the network module
  - TCP Open, 40
- OSCOMMPATCH, 94
- OSCOMMVERS, 94
- OSINTVERS, 94
- OVERFLOW, 92
- OVERRUN, 92

## P

- Performance
  - TCP Open, 41
- Procedure, 10

## R

- Report
  - Open TCP, 30
- Reports
  - TCP Open, 31
- Retrieves data from a socket
  - FCT\_RECEIVE, 63

## S

- Sending data to a specified socket
  - FCT\_SEND, 71
- Sets the socket options
  - FCT\_SETSOCKOPT, 75
- Socket/IP binding
  - FCT\_BIND, 49
- STRINGERROR, 91

## T

### TCP Open

- Debugging, 42

- Diagnostics, 42

- Operating modes of the network module, 40

- Performance, 41

- Reports, 31

## U

Unconditional FFB Call, 15

UTWPORTADDR, 94

## X

XWAYNETWADDR, 94

