

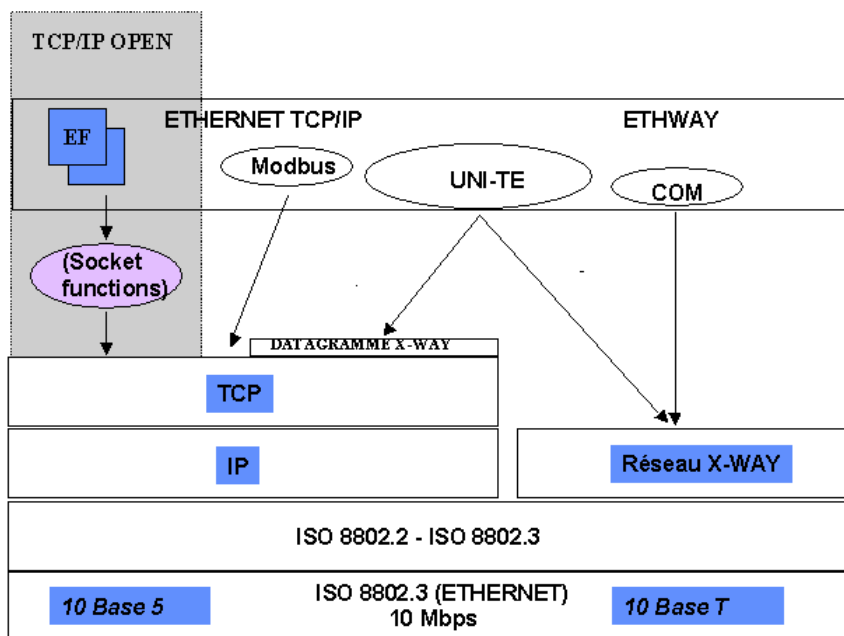


| Section | Page |
|--|------------|
| 1 Presentation | 1/1 |
| 1.1 Introduction | 1/1 |
| 2 Install | 2/1 |
| 2.1 Install the TCP/IP library | 2/1 |
| 2.2 Install the EF of the OPEN TCP kit | 2/1 |
| 2.2-1 Install the executable files | 2/1 |
| 2.2-2 Install the sources files | 2/1 |
| 2.3 Introduction | 2/2 |
| 2.4 Sockets usage and management overview | 2/3 |
| 2.5 General structure of a TCP/IP communication function | 2/4 |
| 2.6 Socket() function | 2/8 |
| 2.7 bind() function | 2/10 |
| 2.8 listen() function | 2/11 |
| 2.9 accept() function | 2/12 |
| 2.10 setsockopt() function | 2/14 |
| 2.11 select() function | 2/16 |
| 2.12 send() function | 2/18 |
| 2.13 recv() function | 2/19 |

| Section | Page |
|--|------------|
| 2.14 shutdown() function | 2/21 |
| 2.15 close() function | 2/22 |
| 3 EF level 1 for advanced programming | 3/1 |
| 4 Operating modes | 4/1 |
| 5 Diagnostic | 5/1 |
| 6 Performance | 6/1 |
| 6.1 Number of connections | 6/1 |
| 6.2 Exchange data performances | 6/1 |
| 7 Application notes | 7/1 |
| 7.1 Elementary function level 1 for advanced programming | 7/1 |
| 7.1.-1 EF : FCT_SOCKET_DFB | 7/1 |
| 7.1.-2 EF : FCT_BIND_DFB | 7/3 |
| 7.1.-3 EF : FCT_CONNECT_DFB | 7/5 |
| 7.1.-4 EF : FCT_LISTEN_DFB | 7/6 |
| 7.1.-5 EF : FCT_ACCEPT_DFB | 7/8 |
| 7.1.-6 EF : FCT_SEND_DFB | 7/10 |
| 7.1.-7 EF : FCT_RECEIVE_DFB | 7/12 |
| 7.1.-8 EF : FCT_STSCOPT_DFB | 7/14 |
| 7.1.-9 EF : FCT_SELECT_DFB | 7/16 |
| 7.1.-10 EF : FCT_SHUTDOWN_DFB | 7/18 |
| 7.1.-11 EF : FCT_CLOSE_DFB | 7/20 |
| 7.2 Client-server model | 7/22 |
| 7.3 Client-server : operating modes | 7/23 |
| 8 Glossary of terms | 8/1 |

1.1 Introduction

OPEN TCP for TSX PREMIUM is a set of components providing facilities to make TCP/IP communication Elementary Functions (EF) for a TSX Premium application.



OPEN TCP Product contents:

OPEN-TCP is based on a new evolution (V2.8 and further) of the ETY5102 module a separate CD ROM after the setup the PL7 directory has the following directories:

- Directory PL7\OFLIB32\LibTcp\ TCP/IP library to add to PL7 SDKC tool. It's based on a simple subset of Berkeley sockets API and provides functions to make a TCP/IP connection server and data exchanges.
- Directory PL7\OFLIB32\ The technical specification (this document)
- Directory EFL1-adv\ Another subset (executable files) of Elementary functions level1 with a specific interface adapted to advanced programming.
- Directory PL7\OFLIB32 An example of a Telnet server model including a PL7 application.
- Directory PL7\OFLIB32\Dvt_ef\ which contains the following directory:
- Directory PL7\OFLIB32\Fam_fffc\ Source files of Elementary function for advanced programming family

The following development tools are needed but not included:

- PL7 Pro environment Version 3.3 IE 72 and further.

The following development tool is needed to develop new EF but is not included:

- PL7 SDKC V3.3 IE 18 and further which allows C programmer to make EF in the PL7 environment.

The required version of the TSX Premium is:

- V3.3 and further

PLEASE NOTE

To use the library it is necessary to have a minimum TCP and sockets knowledge and respect the rules for EF implementation with the SDK C tool. The client and server require a well-known set of conventions before service may be rendered and accepted. This set of convention comprises a protocol which must be implemented at both end of a connection.

The user is in charge of developing EF with the provided tools and manage the operating modes.

Private profile (TCP/IP services, ETHWAY profile, Common Words Exchanges, SNMP) are available but their performances can be impacted depending of the utilization of the OPEN TCP library.

OPEN TCP should be serviced only by qualified sockets knowledge personnel and this document should not be viewed as sufficient instruction for those who are not otherwise qualified to program with the SDK tool. Although reasonable care has been taken to provide accurate and authoritative information in this document, no responsibility is assumed by Schneider Automation for any consequences arising out of the use of this document.

Limits of Schneider Automation responsibilities:

Schneider Automation is not liable for:

- The design and validation of the communication system architecture (client/server operating modes and protocols, performances ..)
- The implementation of appropriate EF (or reuse of example EF included in the OPEN TCP kit)

The test and the validation of EF integrated inside the communication system architecture

- The maintenance and default diagnoses

Under no circumstances shall Schneider Automation assume liabilities for the loss or damage to property, or injury or death of a person, arising out of or resulting from a misuse of the products or from an alteration of the products by the Customer.

2.1 Install the TCP/IP library

With PL7SDKC tool, if you want to create your own Elementary functions using TCP/IP library, you need to install TCPIP library.

Before installation of TCP/IP library, it is necessary to have installed the PL7 SDKC tool on the development system.

To install the library, put the files option.txt and tcpip.lib in the Sdkcefxy/lib directory, overwrite the file sysSDKC3.lib in the Sdkcefxy/lib and put the file tcpip.h in the Sdkcefxy/inc directory. Where xy is the version number of PL7 SDKC which must be at least version 3.3 IE 18.

Now the PL7 SDKC is ready to accept TCP/IP developments.

2.2 Install the EF of the OPEN TCP kit

2.2.1 Install the executable files :

With PL7 programming tool, if you want to use the Elementary functions of the OPEN TCP product, you need to install the executable files:

To install these executable files, you don't need the PL7 SDKC tool.

Elementary function level 1 for advanced programming

On the PC, insert the CD ROM in the drive and select the DFBs and EFs setup.

2.2.2 Install the source files:

With PL7SDKC tool, if you want to visualize or modify the Elementary functions of the OPEN TCP product, you need to install the source files after the TCP/IP library installation.

Before installation of source files, it is necessary to have installed the PL7 SDKC tool on the development system.

It is necessary to copy the following directories into the directory Dvt_ef\ of the "EF Family general directory" of the PL7 SDKC tool:

- Directory Fam_fffc\ Source files of Elementary function for advanced programming Family

2.3 Introduction

All TCP/IP communications functions are executed in an EF context, then they are non blocking functions. Each call to a library function initiate a transaction with the Ethernet module ETY5102 which is the effective executor of the service. The transaction is started at the end of the PLC cycle. Then the programmer must test the activity bit which indicates the end of the function.

The number and the possibilities of the provided socket functions are voluntary reduced. Some parameters and options of the socket usage are forced by the ETY5102 module. See the **socket()** function description.

It is possible to call more than one TCP/IP service in the same PLC cycle but it is not guaranteed that the treatments will be executed in the same order than the calls. **So it is important to wait for execution of a function to ask for a new service on the same socket** (Ex : wait **socket()** return before asking **bind()**, wait **bind()** return before asking **listen()**,...). For data exchange (**send()** and **recv()** functions), **the responsibility of ordering messages is to the user.**

Rules for EF implementation:

It is not possible to make more than one EF per class. Adhering to this rule the PLC system is able to process the operation.

Any addition of a new EF without changing the class number with the SDK C involves system failure and application destruction.

2.4 Sockets usage and management overview

A socket is an endpoint of communication. It is the basic building block for communication. Communication are executed by sending and receiving data through sockets. The TCP/IP library provides only stream sockets using TCP and providing a connection-based communication service.

In the TSX premium architecture, the sockets descriptors consists of an number from 1 to 32.

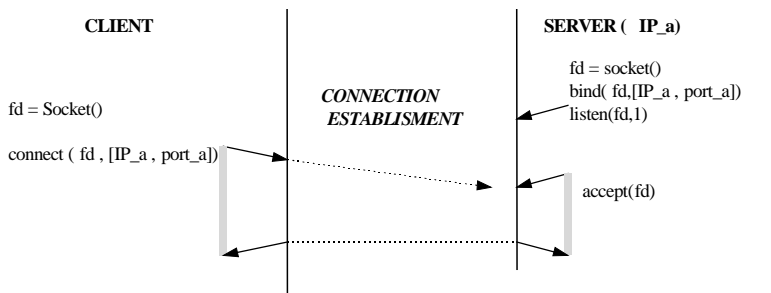
Numbers 1 from 16 are affected to sockets created by the `socket` function. They are listening sockets. Numbers 17 to 32 are affected to sockets created by the `accept` function. They are connected sockets.

Sockets are created via the **`socket()`** function. A socket number is returned, which is then used by the creator to access the socket.

Sockets are created without addresses (IP address and port number). Until a port is bound to a socket, it cannot be used to receive data. The **`bind()`** function is used to bind a port number to a socket. **`bind()`** creates an association between the socket and the port number specified. The IP address is bound at the same time directly by ETY5102 with its local IP address.

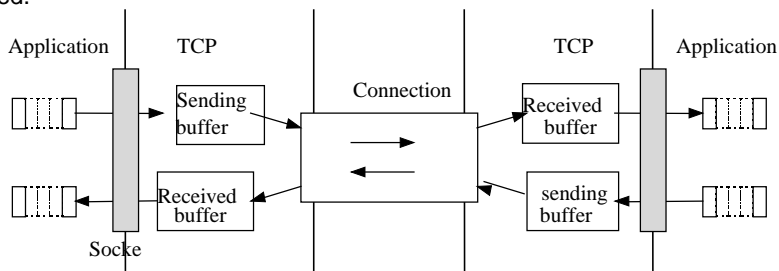
The PLC is always server (for TCP connections), so it must bind a port to the its socket and then use the **`listen()`** function to set up the socket. Then the socket can accept connection requests from clients.

In order to complete a connection, the server must issue the **`accept()`** function specifying the socket number that was specified in the prior **`listen()`** call. A new socket is created with the same properties as the initial one. This new socket is connected to the client's socket, and its number is returned to the server. The initial socket is thereby free for other clients that might want connect with the server.



Connection Establishment

After a connection is established, data can be transferred. The **send()** and **recv()** functions are designed specifically for use with sockets that are already connected.



Data exchanges on TCP connection

The **setsockopt()** function allows a socket's creator to associate options with a socket. These options modify the behavior of the socket.

The **select()** function allows the programmer to test events on all sockets.

The **shutdown()** function allows a socket user to disable sends and/or receives on the socket.

Once a socket is no longer needed, its socket descriptor can be discarded by using the **close()** function.

2.5 General structure of a TCP/IP communication function

The processing of TCP/IP communication functions is asynchronous in relation to the processing of the application task used to activate them. A function is said asynchronous when it is executed during one or more PLC tasks after that which activated it.

A TCP/IP communication function uses :

- An interface number
- parameters specific to the function.
- management parameters.

Each function is structured as follows:

Function (Interface number, Specific parameters, Management parameters)

Interface number :

This parameter is the physical position of the ETY5102 module in the main rack (only the rack number 0 is available).

Specific parameters :

These parameters are specific to each type of TCP/IP communication function. They are described in the sections dedicated to these individual functions.

Management parameters :

The management parameters are common to all TCP/IP communication functions. They comprise:

- a parameter which gives information on the function activity,
- a parameter which contains the exchange report (system report and function report),
- a length parameter for storing the number of bytes to be sent (function **send()**) or the number of bytes received (function **recv()**),

These parameters needs a table of four consecutive words with the following structure :

| | Word number | byte 1 | byte 0 |
|----------------------------|-------------|-----------------|----------------------|
| Data managed by the system | 1 | System reserved | bit 0 : activity bit |
| | 2 | Functionreport | system report |
| | 3 | System reserved | |
| Data managed by the user | 4 | Length | |

Activity bit:

This bit signals the state of execution of the function. It is set to 1 at the start of execution and returns to 0 at the end.

System report:

The system report represents the result of the exchange between application and ETY5102 module. It is not significant of TCP/IP stack access.

This report is common to all functions. It is significant when the value of the activity bit changes from 1 to 0. The various values of this report are shown in the following table :

| Value | System report |
|-------|--|
| 16#00 | Correct exchange |
| 16#01 | Exchange stopped on time-out |
| 16#0B | No CPU system ressources |
| 16#07 | Network module missing (probably incorrect interface number) |
| 16#FF | Communication with ETY5102 error |

Function report:

This report byte defines the result of the operation on the TCP/IP stack of the ETY5102 module. It is only significant if the system report has the 16#00 or 16#FF value.

If the system report is 16#00, the function report is specific to each function. It is described in the section dedicated to the individual function.

If the system report is 16#FF (refused message), the various values of this functionreport are shown in the following table:

| Value | Function report |
|-------|--|
| 16#0B | System ressource missing (too many EF in the same PLC cycle) |
| 16#0C | ETY5102 not running |

Length:

The length parameter is significant for the user only for **send()** and **recv()** functions.

This field can be modified by the system in all TCP/IP communication functions. So before calling a new **send()** or **recv()** function, programmer must check this parameter, even if it is the same as the precedent call.

Important:

While the function is processing (activity bit=1) the management parameters can't be modified by the application task.

Because of the asynchronous mechanism, these words must be taken in a static memory space (Ex : MW space).

It is recommended that the function reports are always tested as soon as they have been executed and before they are next activated.

On a cold start, it is essential to check that all the management parameters are reset to 0.

Management structure definition

```
typedef struct {  
    unsigned char Activity ;  
    unsigned char reserved1;  
    unsigned char SystemReport ;  
    unsigned char FunctionReport ;  
    unsigned short reserved2 ;  
    unsigned short Length ;  
} Mngt ;
```

Socket definition

```
unsigned short Sock ;
```

2.6 Socket() function

Syntax

```
#include <tcpip.h>
```

```
void socket(unsigned short Interface, Sock far *pSock, Mngt far *pMngt)
```

Description

The socket() function creates a new socket and returns its socket number. The socket is a endpoint of TCP/IP communication. It is created as a STREAM TCP socket with following options :

SO_LINGER with no time out

The SO_LINGER option controls the action taken when unsent data is queued on a socket and a **close()** is performed. See **close()** for a description of the way the SO_LINGER settings affect the semantics of **close()**.

NO_DELAY

Disables delay acknowledgment algorithm. Data is sent immediately over the network instead of waiting for the window to be full.

KEEP_ALIVE

Maintain a connection by periodically transmitting a packet over socket wich is not embedded inside the user TCP data flow.

REUSEADDR

The local port can be reused in a bind() call.

Input

Interface

Specifies the ETY5102 module number

Sock

One word array pointer for socket number.

Mngt

Four words array pointer for management

Output

| | |
|---------------------|--|
| Mngt.SystemReport | error if different from 0 (See error codes ch 2.5) |
| Mngt.FunctionReport | NO_ERROR (0) ENOBUFS (0x37) : The maximum number of socket is reached. |
| Sock | If no error the socket number. |

2.7 bind() function

Syntax

```
#include <tcpip.h>
```

```
void bind(unsigned short Interface, unsigned short SocketNumber, unsigned short
PortNumber, Mngt far *pMngt)
```

Description

This function is used to assign (or bind) an internet address and a port number to a socket. A socket is created without an address and cannot be used to receive data (Ex connection request) until it is assigned one.

The internet address is fixed by ETY5102 to its local configured IP address.

Some Port numbers are not allowed to the user because they are already use by ETY5102. This 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 Automation port), 5000 and 5001 (ETY5102 specifics ports), 1024... (TCP USER ports), 3124... (IO Scanner ports), 7400-8400 (RTPS Ports).

Input

| | |
|--------------|--|
| Interface | Specifies the ETY5102 module number |
| SocketNumber | The socket number |
| PortNumber | Port number to assign to the socket |
| Mngt | Four words array pointer for management. |

Output

| | |
|---------------------|---|
| Mngt.SystemReport | error if different from 0 (See error codes ch 2.5) |
| Mngt.FunctionReport | NO_ERROR (0) |
| | EBADS (0x09) : The socket number is invalid |
| | EADDRINUSE (0x30) : The specified port is already in use. |
| | EADDRNOTAVAIL (0x37) : The port number is not available |
| | EINVALID (0x16) : The socket is already binded |

2.8 listen() function

Syntax

```
#include <tcpip.h>
```

```
void listen(unsigned short Interface, unsigned short SocketNumber, Mngt far *pMngt)
```

Description

This function sets up the specified socket to receive connections. Connection requests are queued on the socket until they are accepted with the accept() call. The length of the queue is fixed to 16. If a connection request arrives while the queue is full, the requesting client gets an ECONNREFUSED error.

Input

| | |
|--------------|--|
| Interface | Specifies the ETY5102 module number |
| SocketNumber | The socket number |
| Mngt | Four words array pointer for management. |

Output

| | |
|---------------------|--|
| Mngt.SystemReport | error if different from 0 (See error codes ch 2.5) |
| Mngt.FunctionReport | NO_ERROR (0) EBADS (0x09) : The socket number is invalid |

2.9 accept() function

Syntax

```
#include <tcpip.h>
```

```
void accept(unsigned short Interface, unsigned short SocketNumber, unsigned  
short far *ClientAddr, Mngt far *pMngt)
```

Description

This function is used to accept a connection request that the specified socket receives from a foreign socket.

Before accept() is called, the socket must be set up to receive a connection request by issuing the listen() call. accept() extracts the first connection request on 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 no pending connections exists on the queue accept() returns an error.

Accept() stores the number of the connected socket in the specified array ClientAddr

Accept() stores the client address of the connected socket in the specified array ClientAddr

Input

| | |
|--------------|--|
| Interface | Specifies the ETY5102 module number |
| SocketNumber | The socket number |
| ClientAddr | Four words array pointer where to store client address |
| Mngt | Four words array pointer for management. |

Output

| | |
|---------------------|--|
| Mngt.SystemReport | error if different from 0 (See error codes ch 2.5) |
| Mngt.FunctionReport | NO_ERROR (0) |
| | EBADS (0x09) : The socket number is invalid |
| | EWOULDBLOCK (0x23) : no connection request. |
| | EINVALID (0x16) : Before accept(), the listen() must be called |
| ClientAddr[0] | If no error the connected socket number.(word) |
| ClientAddr[1] | If no error the Client port number.(word) |
| ClientAddr[2] | If no error the Low order word client IP address. |
| ClientAddr[3] | If no error the High order word client IP address. |

2.10 setsockopt() function

Syntax

```
#include <tcpip.h>
```

```
void setsockopt(unsigned short Interface, unsigned short SocketNumber,
unsigned short Option, Mngt far *pMngt)
```

Description

The setsockopt() function sets options associated with the specified socket. The set of options is voluntarily restricted due to the operating mode of ETY5102. Some options are set at the socket creation. See socket() function description.

Input

| | |
|--------------|---|
| Interface | Specifies the ETY5102 module number |
| SocketNumber | The socket number |
| Option | Specifies the option to be set or reset : DONT_ROUTE : Indicates that the outgoing data should not be routed. Packets directed at unconnected nodes are dropped. RESET_DONT_ROUTE :reset DONT_ROUTE . KEEP_ALIVE : Maintains a connection by periodically transmitting a packet over socket. RESET_ KEEP_ALIVE : reset KEEP_ALIVE . |
| Mngt | Four words array pointer for management. |

Output

Mngt.SystemReport

error if different from 0 (See error codes ch 2.5)

Mngt.FunctionReport

NO_ERROR (0)
EBADS (0x09) : The socket number is invalid
EINVAL (0x16) : Invalid option.

Mngt.Length

Number of bytes stored in the buffer if no error.

| Option | Value |
|------------------|-------|
| DONT_ROUTE | 1 |
| RESET_DONT_ROUTE | 2 |
| KEEP_ALIVE | 3 |
| RESET_KEEP_ALIVE | 4 |

2.11 select() function

Syntax

```
#include <tcpip.h>
```

```
void select(unsigned short Interface, unsigned short far *pMask, Mngt far *pMngt)
```

Description

The select() function is used to multiplex I/O requests among multiple sockets. A bit mask of two words is returned by select() function which indicates which of them have available events to treat.

In the TSX premium architecture, the sockets descriptors consists of an number from 1 to 32.

Numbers 1 from 16 are affected to sockets created by the socket function. They are listening sockets. Numbers 17 to 32 are affected to sockets created by the accept function. They are connected sockets. Then the first word of the mask is a bit array of listening sockets (bit 0 is socket 0) and the second word is a bit array of connected sockets.

Input

| | |
|-----------|--|
| Interface | Specifies the ETY5102 module number |
| pMask | Two words array pointer to return the mask of sockets. |
| Mngt | Four words array pointer for management. |

Output

| | |
|---------------------|---|
| Mngt.SystemReport | error if different from 0 (See error codes ch 2.5) |
| Mngt.FunctionReport | NO_ERROR (0) |

pMask

Two words array pointer to return the mask of sockets. Each bit set to 1 shows a event on the socket corresponding of the bit number.

Example :

If bit 3 of the second word is set to 1 the socket number 20 is to be read by the `recv()` function.

If bit 5 of the first word is set 1 the socket number 6 is to read by the `accept()` function.

If the bit is set on a listening socket, the meaning is:

- a connection request is pending.

If the bit is set on a connected socket, the meaning is:

- datas to read on the socket.
- The connection is broken.

2.12 send() function

Syntax

```
#include <tcpip.h>
```

```
void send(unsigned short Interface, unsigned short SocketNumber, char far *pBuf,
Mngt far *pMngt)
```

Description

The send() function is used to send data to a foreign socket. The maximum data length to send is 240 bytes

It is not allowed to send out of band data.

Input

| | |
|--------------|---|
| Interface | Specifies the ETY5102 module number |
| SocketNumber | The socket number |
| pBuf | Points to a buffer containing data to send |
| Mngt | Four words array pointer for management. |
| Mngt.Length | Number of bytes to send. The maximum length is 240 bytes. |

Output

| | |
|---------------------|---|
| Mngt.SystemReport | error if different from 0 (See error codes ch 2.5) |
| Mngt.FunctionReport | NO_ERROR (0) EBADS (0x09) : The socket number is invalid EPIPE (0x20) : The connection is broken EWOULDBLOCK (0x23) : The socket is full. ECONNRESET (0x36) : The connection has been reset by peer. ENOTCONN (0x39) : The socket is not connected (listening socket). INCORRECTLENGTH (0x0E) ; The Length > 240 |
| Mngt.Length | Number of bytes sent if no error. |

2.13 recv() function

Syntax

```
#include <tcpip.h>
```

```
void recv(unsigned short Interface, unsigned short SocketNumber, char far *pBuf,
Mngt far *pMngt)
```

Description

The recv() function is used to receive data from the specified socket. It copies whatever data is available at the socket to the user buffer and returns. The maximum length of data to read is 240 bytes.

The recv() function returns the number of bytes received, and this value should always be checked because this is the only way to detect the actual number of data bytes stored in the user buffer.

It is not allowed to receive out of band data.

Input

| | |
|--------------|---|
| Interface | Specifies the ETY5102 module number |
| SocketNumber | The socket number |
| pBuf | Points to a buffer where data is stored. |
| Mngt | Four words array pointer for management. |
| Mngt.Length | Specifies the size in bytes of the buffer. The maximum length is 240 bytes. |

Output

| | |
|---------------------|--|
| Mngt.SystemReport | error if different from 0 (See error codes ch 2.5) |
| Mngt.FunctionReport | NO_ERROR (0) |
| | EBADS (0x09) : The socket number is invalid |
| | EWOULDBLOCK (0x23) : No data to read. |
| | ECONNRESET (0x36) : The connection has been reset by peer. |
| | ENOTCONN (0x39) : The socket is not connected (listening socket) |

INCORRECTLENGTH (0x0E) ; The Length > 240

ETIMEDOUT (0x3C) : The keep alive timed out on broken connexion.

Mngt.Length

Number of bytes stored in the buffer if no error.

2.14 shutdown() function

Syntax

```
#include <tcpip.h>
```

```
void shutdown(unsigned short Interface, unsigned short SocketNumber, unsigned short how, Mngt far *pMngt)
```

Description

Call this function to disable sends and/or receives on the socket. **shutdown()** is used to disable reception, transmission, or both. If how is 0, subsequent receives on the socket will be disallowed.

The TCP window is not changed and incoming data will be accepted (but not acknowledged) until the window is exhausted. If how is 1, subsequent sends are disallowed. A FIN will be sent. Setting how to 2 disables both sends and receives as described above.

Note that **shutdown()** does not close the socket, and resources attached to the socket will not be freed until **close()** is called. An application should not rely on being able to reuse a socket after it has been shut down.

Input

| | |
|--------------|---|
| Interface | Specifies the ETY5102 module number |
| SocketNumber | The socket number |
| how | specifies the shutdown mechanism. Three options are available as follows : 0 no further receives are allowed on the socket 1 no further sends are allowed on the socket 2 no further sends or receives are allowed on the socket |
| Mngt | Four words array pointer for management. |

Output

| | |
|---------------------|--|
| Mngt.SystemReport | error if different from 0 (See error codes ch 2.5) |
| Mngt.FunctionReport | NO_ERROR (0) EBADS (0x09) : The socket number is invalid EINVAL (0x16) : An argument is not valid. ENOTCONN (0x39) : The socket is not connected.. |

2.15 close() function

Syntax

```
#include <tcpip.h>
```

```
void close(unsigned short Interface, unsigned short SocketNumber, Mngt far *pMngt)
```

Description

The **close()** function delete the specified socket.

Since SO_LINGER is set with a zero timeout interval, **close()** is not blocked even if queued data has not yet been sent or 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 **recv()** call on the remote side of the circuit will fail with CONNRESET.

Input

| | |
|--------------|--|
| Interface | Specifies the ETY5102 module number |
| SocketNumber | The socket number |
| Mngt | Four words array pointer for management. |

Output

| | |
|---------------------|---|
| Mngt.SystemReport | error if different from 0 (See error codes ch 2.5) |
| Mngt.FunctionReport | NO_ERROR (0) EINVALID (0x16) : The socket number is invalid |

Important : If the SocketNumber equals 0, the function delete **all the sockets** of the OPEN profile.

EF level 1 for advanced programming

Size of the EF generated in bytes:

| EF name | Size of EF in bytes |
|--------------------|----------------------------|
| FCT_SEND_DFB | 610 |
| FCT_RECEIVE_DFB | 626 |
| FCT_SOCKET_DFB | 640 |
| FCT_BIND_DFB | 862 |
| FCT_LISTEN_DFB | 848 |
| FCT_ACCEPT_DFB | 880 |
| FCT_SETSOCKOPT_DFB | 862 |
| FCT_SELECT_DFB | 868 |
| FCT_SHUTDOWN_DFB | 862 |
| FCT_CLOSE_DFB | 848 |
| FCT_CONNECT_DFB | 920 |

The ETY5102 module has 4 operating states which are power off, self-test phase, not configured and configured. After power-up, the module performs its self-tests. The module does not operate with a default configuration. The configuration must be transmitted by the PL7 application of the local PLC. When the operating state is not the configured one, the module is not able to receive socket operations then a refuse message is returned to the socket library

The configuration is transmitted by the PL7 application in the following operating modes:

- On New application download.
- After power-up or module disconnection from rack.
- After warm restart
- On Premium reset

After receiving the configuration, the module resets the communication in progress before configuring itself (exchanges in progress are ended, TCP opened connections are closed and all the sockets discarded).

Then the programmer has to check for the S0 and S1 system bit in his application program and create the connections again if a warm or cold restart is detected.

The run to stop and stop to run transitions have no effect on the sockets states.

It is to the programmer to check the connection states (when there are closed). A client which has terminate a connection without the good TCP sequence (Ex : power off) is seen as connected until a send() operation or some other events (Ex : KEEP_ALIVE sequence). A recv call has no effect on the net and can't cause a disconnection detection (see application notes).

The number of connections in the PL7 debugging window includes private and open profiles connections.

The IP addresses of clients of the open profile can't be seen in this window.

IP communication test

It is possible to use the communication test window for testing the IP communication with client equipment if the IP address of the client is declared in the grip of remote devices (used by the private profile). The list of configured IP addresses 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.

6.1 Number of connections

The maximum number of connections allowed for OPEN TCP/IP use is 16. This number can be less than 16 if the private TCP/IP profile use more than 16 connections because the total number of connections is maximum 32 for ETY5102 and the private TCP/IP profile can use until 32 connections depending of the configuration.

6.2 Exchange data performances

TCP/IP connection is a byte data flow. Information can be append to this data flow at any time. Due to X-Bus transport mechanism limitation, the maximum chunk of data is limited to 240 bytes per PLC cycle if the user want to keep the sequential order of information transmission. A 8 Kbytes message takes 35 PLC cycles ($8 \times 1024 / 240$) to be sent or received. It corresponds to 1.75 sec with 50ms cycle time. This for one socket.

For message oriented protocol, a lower level interface has to manage the fragmentation process. The performance at this time is depends on the number of send() or receive() executed in the same PLC cycle.

These performances can be reduced depending of the ETY5102 module usage for others communications tasks on Private profile , ETHWAY profile or Common Words Exchanges.

7.1 Elementary function level 1 for advanced programming

7.1.1 EF : FCT_SOCKET_DFB

7.1.1.1 Description

The EF FCT_SOCKET is an encapsulation of the socket() function in order to use the function from the PL7 language. It creates a new socket and returns its socket number.

7.1.1.2 SDKC source function

```
//PL7SDK_BEGIN_FUNCTION_NAME
// do not modify the generated function prototype please
// The main EF Function must be the first in this file
#include <Cstsys.h>
#include <Fctsys.h>
#include <C:\ASAWTEMP\SDKC\CLASSE03.H >
#include <C:\ASAWTEMP\SDKC\MAIN0301.H>
void far pascal FCT_SOCKET_DFB(
short INTE, //Module number
short IGST, //Index of the first management word in the
table GEST
adrTABLE SOCK, //Returned Socket number
adrTABLE GEST //Management parameters
)
//PL7SDK_END_FUNCTION_NAME
{
    #include <TcpIp.h>

    Mngt far *pParametreGestion;
    unsigned short far *pSocket;

    /* Physical address of the management table address
    calculation */
    pParametreGestion = (Mngt *)GET_ADDR (GEST.selecteur,
    GEST.offset);
    pSocket = (unsigned short *)GET_ADDR (SOCK.selecteur,
    SOCK.offset);

    socket((unsigned short)INTE,
    pSocket, p [IGST]);
}
```

7.1.1.3 Interface

Input Parameters :

| Name | Type | Comment |
|------|------|--|
| INTE | WORD | Module number |
| IGST | WORD | Index of the first management word in the table GEST |

Output Parameters :

| Name | Type | Comment |
|------|------|------------------------|
| SOCK | WORD | Returned Socket number |

I/O Parameters :

| Name | Type | Comment |
|------|------|----------------------|
| GEST | AR_W | Management parameter |

7.1.1.4 PL7 programming

Into the ST language , this EF will be called like this :

(* open socket on module number 4 *)

FCT_SOCKET_DFB(4,%MW10:1,%MW20:4);

7.1.2 EF : FCT_BIND_DFB

7.1.2.1 Description

The EF FCT_BIND is an encapsulation of the bind() function in order to use the function from the PL7 language. This EF is used to assign (or bind) an internet address and a port number to a socket.

7.1.2.2 SDKC source function

```
//PL7SDK_BEGIN_FUNCTION_NAME
// do not modify the generated function prototype please
// The main EF Function must be the first in this file
#include <Cstsys.h>
#include <Fctsys.h>
#include <C:\ASAWTEMP\SDKC\CLASSE04.H >
#include <C:\ASAWTEMP\SDKC\MAIN0401.H>
void far pascal FCT_BIND(
short INTE, //Module number
short SOCK, //Socket number
short PORT, //Port number
short IGST, //Index of the first management word in the
table GEST
adrTABLE GEST //Management parameters
)
//PL7SDK_END_FUNCTION_NAME
{
    #include <TcpIp.h>
    Mngt far *pParametreGestion;

    /* Physical address of the sending buffer address calculation
    */
    pParametreGestion = (Mngt *)GET_ADDR (GEST.selecteur,
    GEST.offset);

    bind((unsigned short)INTE, (unsigned short)SOCK, (unsigned
    short)PORT, pParametreGestion[IGST]);
}
```

7.1.2.3 Interface

Input Parameters :

| Name | Type | Comment |
|------|------|--|
| INTE | WORD | Module number |
| SOCK | WORD | Socket number |
| PORT | WORD | Port number |
| IGST | WORD | Index of the first management word in the table GEST |

Output Parameters :

| Name | Type | Comment |
|------|------|---------|
| | | |

I/O Parameters :

| Name | Type | Comment |
|------|------|----------------------|
| GEST | AR_W | Management parameter |

7.1.2.4 PL7 programming

Into the ST language , this EF will be called like this :

(*Bind socket number 5 of module number 4 with the port number 505*)

```
FCT_BIND_DFB(4,5,505,%MW0:4);
```

7.1.3 EF : FCT_CONNECT_DFB

7.1.3.1 Description

The EF FCT_CONNECT_DFB is an encapsulation of the connect() function in order to use the function from the PL7 language. This EF is used to establish a connection on a well-known port and internet address.

7.1.3.2 Interface

Input Parameters :

| Name | Type | Comment |
|------|------|--|
| INTE | WORD | Module number |
| SOCK | WORD | Socket number |
| ISRV | WORD | Index of the first management word in the table SERV |
| IGST | WORD | Index of the first management word in the table GEST |

Output Parameters :

| Name | Type | Comment |
|------|------|---------|
| | | |

I/O Parameters :

| Name | Type | Comment |
|------|------|---------------------------------|
| SERV | AR_W | Port + IP address of the server |
| GEST | AR_W | Management parameter |

7.1.3.3 PL7 programming

Into the ST language , this EF will be called like this :

(*Connect socket number 5 of module number 4 with the server *)

FCT_CONNECT_DFB(4,5,0,0,%MW10:3,%MW0:4);

7.1.4 EF : FCT_LISTEN_DFB

7.1.4.1 Description

The EF FCT_LISTEN is an encapsulation of the listen() function in order to use the function from the PL7 language. This EF sets up the specified socket to receive connections.

7.1.4.2 SDKC source function

```
//PL7SDK_BEGIN_FUNCTION_NAME
// do not modify the generated function prototype please
// The main EF Function must be the first in this file
#include <Cstsyst.h>
#include <Fctsyst.h>
#include <C:\ASAWTEMP\SDKC\CLASSE05.H >
#include <C:\ASAWTEMP\SDKC\MAIN0501.H>
void far pascal FCT_LISTEN(
short INTE, //Module number
short SOCK, //Socket number
short IGST, //Index of the first management word in the
table GEST
adrTABLE GE //Management parameters
)
//PL7SDK_END_FUNCTION_NAME
{
    #include <TcpIp.h>
    Mngt far *pParametreGestion;

    /* Physical address of the management table address
    calculation */
    pParametreGestion = (Mngt *)GET_ADDR (GEST.selecteur,
    GEST.offset);

    listen((unsigned short)INTE, (unsigned short)SOCK,
    pParametreGestion[IGST]);
}
```

7.1.4.3 Interface

Input Parameters :

| Name | Type | Comment |
|------|------|--|
| INTE | WORD | Module number |
| SOCK | WORD | Socket number |
| IGST | WORD | Index of the first management word in the table GEST |

Output Parameters :

| Name | Type | Comment |
|------|------|---------|
| | | |

I/O Parameters :

| Name | Type | Comment |
|------|------|----------------------|
| GEST | AR_W | Management parameter |

7.1.4.4 PL7 programming

Into the ST language , this EF will be called like this :

(* Listen socket number %MW10 of module number 4 *)

FCT_LISTEN_DFB(4,%MW10,%MW0:4);

7.1.5 EF : FCT_ACCEPT_DFB

7.1.5.1 Description

The EF FCT_ACCEPT is an encapsulation of the accept() function in order to use the function from the PL7 language. This EF is used to accept a connection request that the specified socket receives from a foreign socket.

7.1.5.2 SDKC source function

```
//PL7SDK_BEGIN_FUNCTION_NAME
// do not modify the generated function prototype please
// The main EF Function must be the first in this file
#include <Cstsys.h>
#include <Fctsys.h>
#include <C:\ASAWTEMP\SDKC\CLASSE06.H >
#include <C:\ASAWTEMP\SDKC\MAIN0601.H>
void far pascal FCT_ACCEPT(
short INTE,//Module number
short SOCK,//Socket number
short ICLIE,//Index of the first word in the table
CLIE
short IGST,//Index of the first management word in the table
GEST
adrTABLE CLIE,//Socket service + port + IP du client
adrTABLE GEST//Management parameter
)
//PL7SDK_END_FUNCTION_NAME
{
    #include <TcpIp.h>

    Mngt far *pParametreGestion;
    unsigned short far *pParametreClient;

    /* Physical address of the management table address
    calculation */
    pParametreClient = (unsigned short *)GET_ADDR
    (CLIE.selecteur, CLIE.offset);
    /* Physical address of the management table calculation */
    pParametreGestion = (Mngt *)GET_ADDR (GEST.selecteur,
    GEST.offset);

    accept((unsignedshort)INTE,(unsigned short)SOCK,
    pParametreClient[ICLIE],pParametreGestion[IGST]);}
```

7.1.5.3 Interface

Input Parameters :

| Name | Type | Comment |
|-------|------|---|
| INTE | WORD | Module number |
| SOCK | WORD | Socket number |
| ICLIE | WORD | Index of the first word in the table CLIE |
| IGST | WORD | Index of the first word in the table GEST |

Output Parameters :

| Name | Type | Comment |
|------|------|-----------------------------------|
| CLIE | AR_W | Socket service + port + IP client |

I/O Parameters :

| Name | Type | Comment |
|------|------|----------------------|
| GEST | AR_W | Management parameter |

7.1.5.4 PL7 programming

Into the ST language , this EF will be called like this :

(* Accept connection on socket %MW10 on module number 4 *)

FCT_ACCEPT_DFB(4,%MW10,%MW20:4,%MW0:4);

7.1.6 EF : FCT_SEND_DFB

7.1.6.1 Description

The EF FCT_SEND is an encapsulation of the send() function in order to use the function from the PL7 language. The function is used to send data to a foreign socket. The maximum length of datas to send is 240 bytes.

7.1.6.2 SDKC source function

```
//PL7SDK_BEGIN_FUNCTION_NAME
// do not modify the generated function prototype please
// The main EF Function must be the first in this file
#include <Cstsys.h>
#include <Fctsys.h>
#include <C:\ASAWTEMP\SDKC\CLASSE01.H >
#include <C:\ASAWTEMP\SDKC\MAIN0101.H>
void far pascal FCT_SEND(
short INTE,//Module number
short SOCK,//Socket number
short IBUF,//Index of the first character in PBUF
short IGST,//Index of the first management word in the table
GEST
adrTABLE PBUF,//Datas to send
adrTABLE GEST//Management parameter
)
//PL7SDK_END_FUNCTION_NAME
{
#include <TcpIp.h>

    char far *pBuffer;
    Mngt far *pParametreGestion;

/* Physical address of the sending buffer address calculation
*/
    pBuffer = (char *)GET_ADDR (PBUF.selecteur, PBUF.offset);
/* Physical address of the management table address
calculation */
    pParametreGestion = (Mngt *)GET_ADDR (GEST.selecteur,
GEST.offset);

    send((unsigned short)INTE,(unsigned short)SOCK,
pBuffer[IBUF], pParametreGestion[IGST]);
}
```

7.1.6.3 Interface

Input Parameters :

| Name | Type | Comment |
|------|------|---|
| INTE | WORD | Module number |
| SOCK | WORD | Socket number |
| IBUF | WORD | Index of the first word in the table PBUF |
| IGST | WORD | Index of the first word in the table GEST |
| PBUF | AR_W | Datas to send |

Output Parameters :

| Name | Type | Comment |
|------|------|---------|
| | | |

I/O Parameters :

| Name | Type | Comment |
|------|------|----------------------|
| GEST | AR_W | Management parameter |

7.1.6.4 PL7 programming

Into the ST language , this EF will be called like this :

(* Send data on socket number %MW10 of module number 4 *)
 FCT_SEND_DFB(4,%MW10,%MW100:120,%MW0:4);

7.1.7 EF : FCT_RECEIVE_DFB

7.1.7.1 Description

The EF FCT_RECEIVE is an encapsulation of the recv() function in order to use the function from the PL7 language. The function is used to receive data from the specified socket. It copies whatever data is available at the socket to the user buffer and returns. The maximum length of data to read is 240 bytes.

7.1.7.2 SDKC source function :

```
//PL7SDK_BEGIN_FUNCTION_NAME
// do not modify the generated function prototype please
// The main EF Function must be the first in this file
#include <Cstsys.h>
#include <Fctsys.h>
#include <C:\ASAWTEMP\SDKC\CLASSE02.H >
#include <C:\ASAWTEMP\SDKC\MAIN0201.H>
void far pascal FCT_RECEIVE(
short INTE, //Module number
short SOCK, //Socket number
short IBUF, //Index of the first character in PBUF
short IGST, //Index of the first management word in the table
GEST
adrTABLE PBUF, //Received datas
adrTABLE GEST //Management pointer
)
//PL7SDK_END_FUNCTION_NAME
{
    #include <TcpIp.h>

    char far *pBuffer;
    Mngt far *pParametreGestion;

    /* Physical address of the sending buffer address calculation
    */
    pBuffer = (char *)GET_ADDR (PBUF.selecteur, PBUF.offset);

    /* Physical address of the management table address
    calculation */
    pParametreGestion = (Mngt *)GET_ADDR (GEST.selecteur,
    GEST.offset);

    recv((unsigned short)INTE, (unsigned short)SOCK,
    pBuffer[IBUF], pParametreGestion[IGST]);
}
```

7.1.7.3 Interface

Input Parameters :

| Name | Type | Comment |
|------|------|---|
| INTE | WORD | Module number |
| IBUF | WORD | Index of the first word in the table PBUF |
| SOCK | WORD | Socket number |
| IGST | WORD | Index of the first word in the table GEST |

Output Parameters :

| Name | Type | Comment |
|------|------|----------------|
| PBUF | AR_W | Received datas |

I/O Parameters :

| Name | Type | Comment |
|------|------|----------------------|
| GEST | AR_W | Management parameter |

7.1.7.4 PL7 programming

Into the ST language , this EF will be called like this :

(* receive datas on socket number %MW30 of module number 4 *)
 FCT_RECEIVE_DFB(4,%MW30,%MW100:120,%MW0:4);

7.1.8 EF : FCT_STCKOPT_DFB

7.1.8.1 Description

The EF FCT_SETSOCKOPT is an encapsulation of the setsockopt() function in order to use the function from the PL7 language. The function sets options associated with the specified socket. The set of options is voluntarily restricted due to the operating mode of ETY5102. Some options are set at the socket creation. See setsockopt() function description.

7.1.8.2 SDKC source function

```
//PL7SDK_BEGIN_FUNCTION_NAME
// do not modify the generated function prototype please
// The main EF Function must be the first in this file
#include <Cstsys.h>
#include <Fctsys.h>
#include <C:\ASAWTEMP\SDKC\CLASSE07.H >
#include <C:\ASAWTEMP\SDKC\MAIN0701.H>
void far pascal FCT_SETSOCKOPT(
short INTE, //Module number
short SOCK, //Socket number
short OPT, //Type of option
short IGST, //Index of the first management word in the
table GEST
adrTABLE GEST //Management parameter
)
//PL7SDK_END_FUNCTION_NAME
{
    #include <TcpIp.h>

    Mngt far *pParametreGestion;

    /* Physical address of the management table address
    calculation */
    pParametreGestion = (Mngt *)GET_ADDR (GEST.selecteur,
GEST.offset);
    setsockopt((unsigned short)INTE, (unsigned short)SOCK,
(unsigned short)OPT, pParametreGestion[IGST]);
}
```

7.1.8.3 Interface

Input Parameters :

| Name | Type | Comment |
|------|------|---|
| INTE | WORD | Module number |
| SOCK | WORD | Socket number |
| OPT | WORD | Type of option |
| IGST | WORD | Index of the first word in the table GEST |

Output Parameters :

| Name | Type | Comment |
|------|------|---------|
| | | |

I/O Parameters :

| Name | Type | Comment |
|------|------|----------------------|
| GEST | AR_W | Management parameter |

7.1.8.4 PL7 programming

Into the ST language , this EF will be called like this :

(* Set socket option DONT_ROUTE on socket number %MW10 of module 4*)

```
%MW20:= DONT_ROUTE;
```

```
FCT_STSCKOPT_DFB(4,%MW10,%MW20,%MW0:4);
```

7.1.9 EF : FCT_SELECT_DFB

7.1.9.1 Description

The EF FCT_SELECT is an encapsulation of the select() function in order to use the function from the PL7 language. The function is used to multiplex I/O requests among multiple sockets. A bit mask of two words is returned by function which indicates which of them have available events to treat.

7.1.9.2 SDKC source function

```
//PL7SDK_BEGIN_FUNCTION_NAME
// do not modify the generated function prototype please
// The main EF Function must be the first in this file
#include <Cstsyst.h>
#include <Fctsyst.h>
#include <C:\ASAWTEMP\SDKC\CLASSE08.H >
#include <C:\ASAWTEMP\SDKC\MAIN0801.H>
void far pascal FCT_SELECT(
short INTE, //Module number
short IMASK, //Index of the first character in MASK
short IGST, //Index of the first management word in the table
GEST
adrTABLE MASK, //Status of each socket
adrTABLE GEST //Management parameter
)
//PL7SDK_END_FUNCTION_NAME
{
    #include <TcpIp.h>

    Mngt far *pParametreGestion;
    unsigned short far *pMask;

    /* Physical address of the management table address
    calculation */
    pParametreGestion = (Mngt *)GET_ADDR (GEST.selecteur,
GEST.offset);
    /* Physical address of the management table address
    calculation */
    pMask = (unsigned short *)GET_ADDR (MASK.selecteur,
MASK.offset);

    select((unsigned short)INTE, pMask[IMASK],
pParametreGestion[IGST]);
}
```

7.1.9.3 Interface

Input Parameters :

| Name | Type | Comment |
|-------|------|---|
| INTE | WORD | Module number |
| IMASK | WORD | Index of the first word in the table MASK |
| IGST | WORD | Index of the first word in the table GEST |

Output Parameters :

| Name | Type | Comment |
|------|------|-----------------------|
| MASK | AR_W | Status of each socket |

I/O Parameters :

| Name | Type | Comment |
|------|------|----------------------|
| GEST | AR_W | Management parameter |

7.1.9.4 PL7 programming

Into the ST language , this EF will be called like this :

(* Select function on module number 4 *)

FCT_SELECT_DFB(4,%MW10:2,%MW0:4);

7.1.10 EF : FCT_SHUTDOWN_DFB

7.1.10.1 Description

The EF FCT_SHUTDOWN is an encapsulation of the shutdown() function in order to use the function from the PL7 language. Call this function to disable sends and/or receives on the socket.

7.1.10.2 SDKC source function

```
//PL7SDK_BEGIN_FUNCTION_NAME
// do not modify the generated function prototype please
// The main EF Function must be the first in this file
#include <Cstsyst.h>
#include <Fctsyst.h>
#include <C:\ASAWTEMP\SDKC\CLASSE09.H >
#include <C:\ASAWTEMP\SDKC\MAIN0901.H>
void far pascal FCT_SHUTDOWN(
short INTE, //Module number
short SOCK, //Socket number
short HOW, //Shutdown option
short IGST, //Index of the first management word in the
table GEST
adrTABLE GEST //Management parameter
)
//PL7SDK_END_FUNCTION_NAME
{
    #include <TcpIp.h>

    Mngt far *pParametreGestion;

    /* Physical address of the management table address
    calculation */
    pParametreGestion = (Mngt *)GET_ADDR (GEST.selecteur,
GEST.offset);

    shutdown((unsigned short)INTE, (unsigned short)SOCK,
(unsigned short)HOW, pParametreGestion[IGST]);
}
```

7.1.10.3 Interface

Input Parameters :

| Name | Type | Comment |
|------|------|--|
| INTE | WORD | Module number |
| SOCK | WORD | Socket numberr |
| HOW | WORD | Shutdown option |
| IGST | WORD | Index of the first word in the table GEST |

Output Parameters :

| Name | Type | Comment |
|------|------|---------|
| | | |

I/O Parameters :

| Name | Type | Comment |
|------|------|----------------------|
| GEST | AR_W | Management parameter |

7.1.10.4 PL7 programming

Into the ST language , this EF will be called like this :

```
(* Shutdown socket number %MW10 of module number 4 *)
%MW20 :=2; (*no further sends or receives allowed on the socket)
FCT_SHUTDOWN_DFB(4,%MW10,%MW20,%MW0:4);
```

7.1.11 EF : FCT_CLOSE_DFB

7.1.11.1 Description

The EF FCT_CLOSE is an encapsulation of the close() function in order to use the function from the PL7 language. The function delete the specified socket. If the SocketNumber equals 0, the function delete **all the sockets** of the OPEN profile.

7.1.11.2 SDKC source function :

```
//PL7SDK_BEGIN_FUNCTION_NAME
// do not modify the generated function prototype please
// The main EF Function must be the first in this file
#include <Cstsys.h>
#include <Fctsys.h>
#include <C:\ASAWTEMP\SDKC\CLASSE0A.H >
#include <C:\ASAWTEMP\SDKC\MAIN0A01.H>
void far pascal FCT_CLOSE(
short INTE, //Module number
short SOCK, //Socket number
short IGST, //Index of the first management word in the
table GEST
adrTABLE GEST //Management parameter
)
//PL7SDK_END_FUNCTION_NAME
{
    #include <TcpIp.h>

    Mngt far *pParametreGestion;

    /* Physical address of the management table address
    calculation */
    pParametreGestion = (Mngt *)GET_ADDR (GEST.selecteur,
GEST.offset);

    close((unsigned short)INTE, (unsigned short)SOCK,
pParametreGestion[IGST]);
}
```

7.1.11.3 Interface

Input Parameters :

| Name | Type | Comment |
|------|------|--|
| INTE | WORD | Module number |
| SOCK | WORD | Socket number |
| IGST | WORD | Index of the first word in the table GEST |

Output Parameters :

| Name | Type | Comment |
|------|------|---------|
| | | |

I/O Parameters :

| Name | Type | Comment |
|------|------|----------------------|
| GEST | AR_W | Management parameter |

7.1.11.4 PL7 programming

Into the ST language , this EF will be called like this :

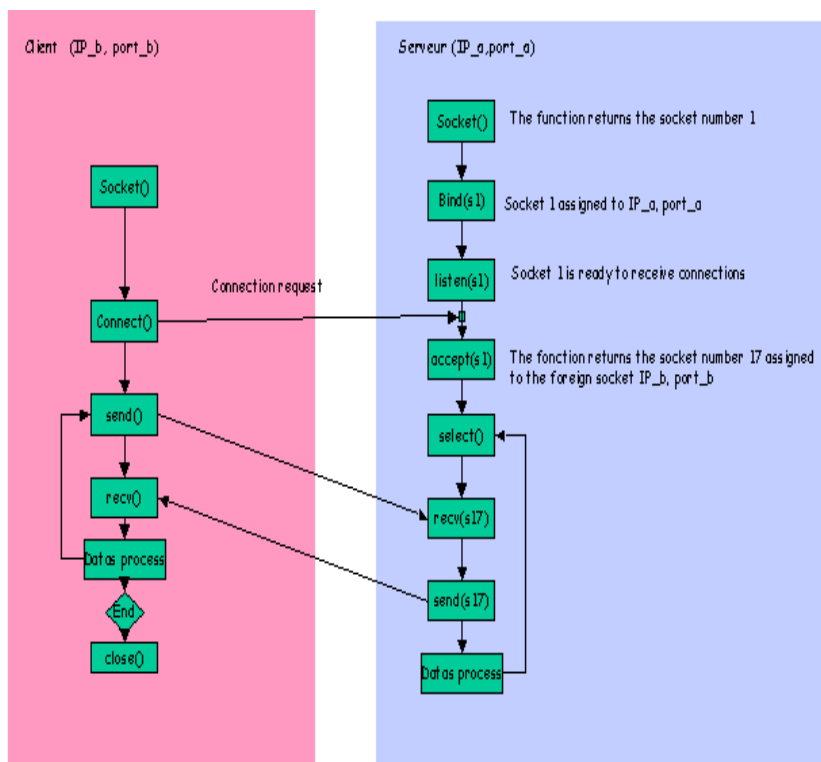
(* Close socket number %MW10 of module number 4 *)

FCT_CLOSE_DFB(4,%MW10,%MW0:4);

7.2 Client-server model

The client and server require a well-known set of conventions before service may be rendered and accepted. This set of convention comprises a protocol which must be implemented at both end of a connection. It is to the programmer to check the connection states.

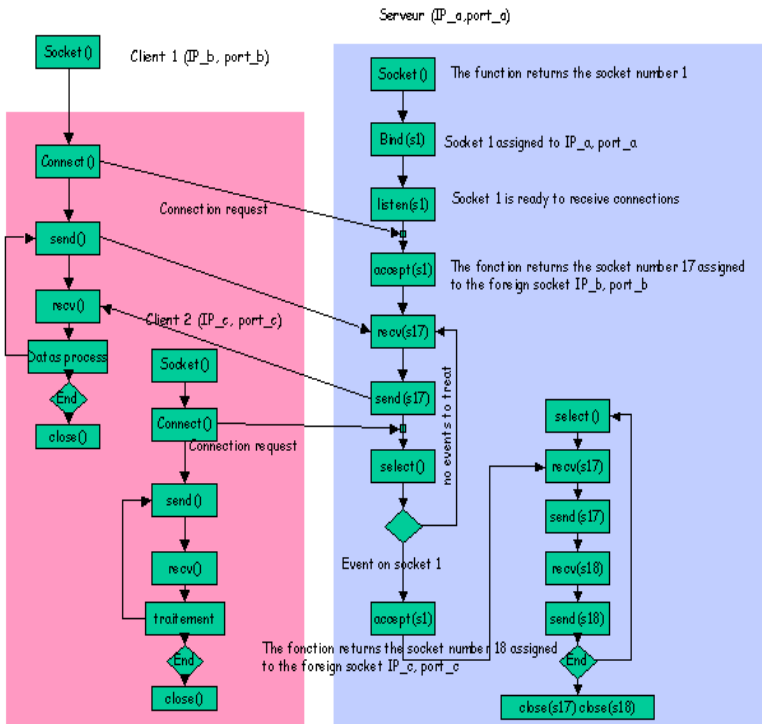
The server application has to listen for service requests. In this scheme client applications request services from a server application. This implies an asymmetry in establishing communication between the client and server.



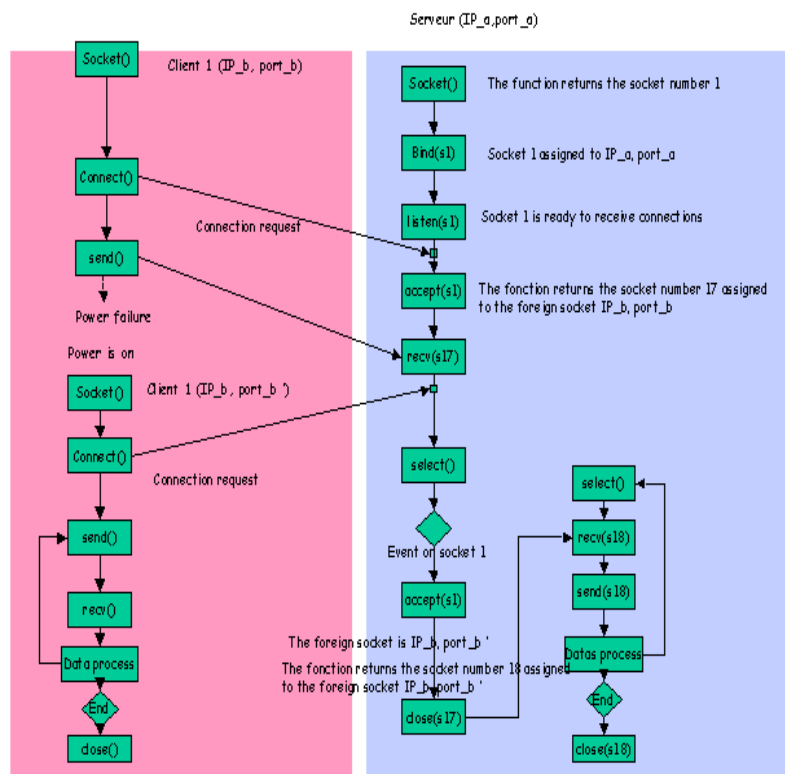
7.3 Client-server : operating modes

It is to the programmer to check the connection states. A client which has terminate a connection without the good TCP sequence (Ex : power off) is seen as connected until a send() operation or some other events (Ex : KEEP_ALIVE sequence). A recv call has no effect on the net and can't cause a disconnection detection.

Example 1 : The server application treats 2 connections requested by 2 clients

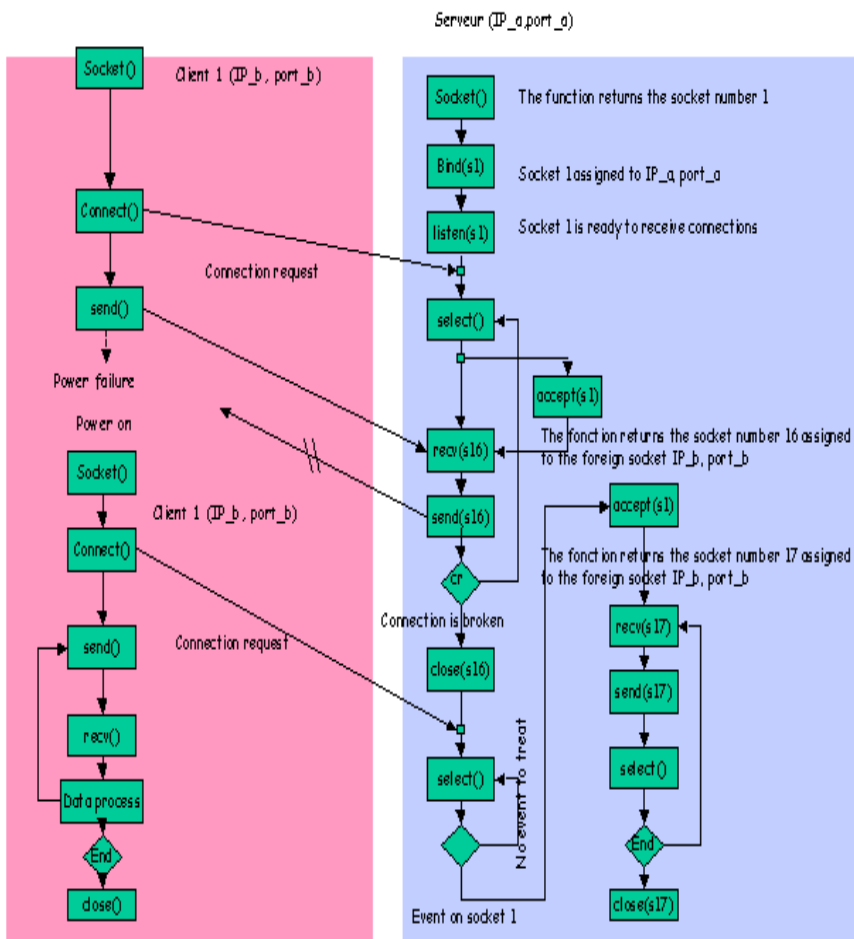


Example 2 : The server application treats 2 connections requested by the same client. The client has terminate a connection without the good TCP sequence (Ex : client power failure).The socket is seen as connected by the server until a new connect() operation is requested by the client.



Please note : If DHCP (Dynamic Host Configuration Protocol) is not used, the server is able to compare the IP addresses of the clients.

Example 3 : The server application treats 2 connections requested by the same client. The client has terminate a connection without the good TCP sequence (Ex : client power failure).The socket is seen as connected by the server until a send() operation.



GLOSSARY OF TERMS

Server:

In TCPIP view , the server is the node which waits for connections.

Client:

In TCPIP view, the client is the node which ask for a connection.

ABBREVIATIONS

EF:Elementary Function.

| Section | Page |
|---|------------|
| 1 General presentation | 1/1 |
| 1.1 General | 1/1 |
| 2 Installation | 2/1 |
| 2.1 Installation procedure | 2/1 |
| 2.2 Configuration of module TSX ETY 110 WS/5102 | 2/2 |
| 2.3 Programming DFBs in an application | 2/3 |
| 2.3-1 Importing DFBs | 2/3 |
| 2.3-2 Create a DFB Instance | 2/3 |
| 3 Installation principles | 3/1 |
| 3.1 TCP connection principles | 3/1 |
| 3.2 Basic principle of DFBs | 3/2 |
| 3.2-1 Establishing connections with access security | 3/2 |
| 3.2-2 Transferring messages | 3/5 |
| 4 Interfaces PL7 application | 4/1 |
| 4.1 Points common to all DFBs | 4/1 |
| 4.1-1 Maintained call | 4/1 |
| 4.1-2 Management table | 4/2 |
| 4.1-3 RST and ACTIVITY bits | 4/4 |
| 4.1-4 ERROR bit - STATUS WORD | 4/5 |
| 5 TCP_CNx DFB connection management | 5/1 |
| 5.1 Presentation | 5/1 |

| Section | Page |
|--|------------|
| 5.1-1 Characteristics | 5/1 |
| 5.1-2 Operation | 5/4 |
| 5.1-3 Programming example - Connection to 1 single TCP port | 5/5 |
| 6 TCP_SEND DFB data transmission | 6/1 |
| 6.1 Presentation | 6/1 |
| 6.1-1 Characteristics | 6/1 |
| 6.1-2 Operation | 6/4 |
| 6.1-3 Programming example | 6/5 |
| 7 TCP_RECEIVE DFB data reception | 7/1 |
| 7.1 Presentation | 7/1 |
| 7.1-1 Characteristics | 7/1 |
| 7.1-2 Operation | 7/4 |
| 7.1-3 Programming example | 7/5 |
| 8 Performances | 8/1 |
| 8.1 Points relevant to the performance of a TCPIP-OPEN communication | 8/1 |
| 8.2 Diagram of EF processing during a PLC cycle | 8/2 |
| 8.3 Performance measurements | 8/2 |
| 9 Appendix | 9/1 |
| 9.1 Differences with OFB PL7-3 | 9/1 |
| 9.2 Reminder on the TCPIP communication | 9/2 |

1.1 General

The TCP communication DFB library used with version 2.8 or later of the TSX ETY 110 WS/5102 module for Premium PLCs is used to transfer blocks of data between a PLC application and a remote application via a TCPIP connection established at the initiative of the remote application.

The remote application is executed on a device that supports the TCPIP communication profile.

The communication DFB library is composed:

- of a connection management TCP_CNX DFB,
- of a TCP_SEND DFB for transmission of data blocks of a maximum size of 8 kilobytes,
- of a TCP_RECEIVE DFB for transmission of data blocks of a maximum size of 8 kilobytes.

Note:

TCP communication DFBs for Premium PLCs use the services from the "OPEN TCP for TSX Premium" range which is itself constituted of an TCP EF (Elementary Function) library: an EF enabling the PL7 application to manage TCP connections and to send and receive byte flows on its connections.

Installation of the product «DFBs of TCPIP free messaging system» completes a PL7 Pro programming workshop of V3.3 or later by installing:

- 3 DFBs: TCP_CNX, TCP_SEND, TCP_RECEIVE,
- and a TCPIP_DFB EF family.

2.1 Installation procedure

The CD-ROM TLXCDUNTCPB33F gives access to the installation procedure and documentation.

Insert the CD-ROM, confirm at the welcome screen and follow the menus.

2.2 Configuration of module TSX ETY 110 WS/5102

The module TSX ETY 110 WS/5102 is configured by PL7 in accordance with the data entered by the user in the module configuration screen:

- Local IP address of the PLC on the network.
- The SubNet Mask.
- The IP address of the gateway by default.
- The use of Ethernet 802.3 frames with SNAP (SubNetwork Access Protocol).

The module TSX ETY 110 WS/5102 shall be installed in rack 0.

2.3 Programming DFBs in an application

2.3-1 Importing DFBs

In "Structure View" of PL7 Pro, under the "DFB types" heading, select "Import Binary".

Import each of the 3 DFBs TCP_CNX, TCP_RECEIVE, TCP_SEND into the application.

2.3-2 Create a DFB Instance

From within PL7 Pro select "Tools", "Library" then the "DFB" thumbnail tab.

Then choose the type of DFB that you wish to instantiate and confirm with "Create".

Next type the instance name of your DFB and confirm with "Create".

Your DFB is now instantiated. The instance name is the one which is used in the PL7 PLC program.

CAUTION

It is only possible to import the DFBs if the installation procedure of the CD-ROM TLXCDUNTCPB33F has been fully executed:

- installation of the TCP_CNX, TCP_RECEIVE, TCP_SEND DFBs,
- installation of the TCPIP_DFB EF family in the library.

3.1 TCP connection principles

The connection is established between the two applications asymmetrically, and is based on a client/server model.

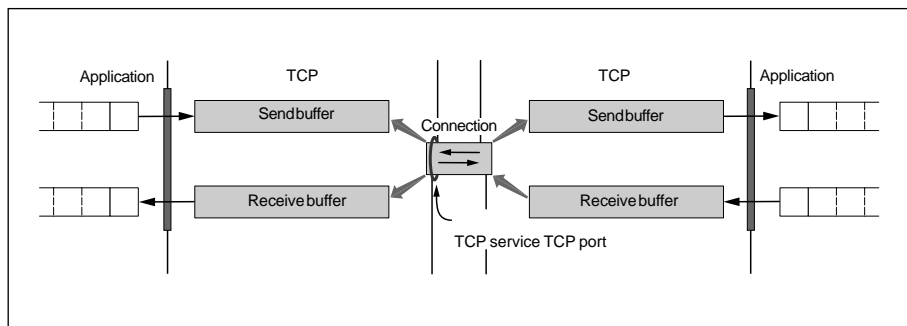
In order for a connection to be established between the two applications, the server application must be awaiting an incoming connection on a TCP service port.

The TCP protocol uses the port numbers to identify destinations on a given machine. When the port number is identified by the server, the client can then send a connection request to the server application.

The server application accepts the incoming connection. Once the connection is open, a send buffer and a receive buffer are attributed to this TCP service port.

Data can be transferred on an established connection.

The data transfer is reliable, (data stored in buffer zone), non-structured (byte and not message flows) and simultaneously bidirectional (full-duplex).



3.2 Basic principle of DFBs

3.2-1 Establishing connections with access security

The server PLC awaits incoming connections for each local TCP service port.

This connection queue is managed by the TCP_CNx DFB. All the remote machines declared in the configuration of the TCP_CNx DFB can then connect to a PLC. Immediately after an incoming connection is accepted on a local service port, the TCP_CNx DFB checks that the IP address of the remote machine is in the list of remote machines with authorization to connect.

If the remote IP address is given in this list, the connection to the local TCP service port is accepted. This connection is then stored in the list of open connections. The associated local TCP service port is busy and is not be accessible for other connection requests.

If the remote IP address is not included in the list of authorized addresses, the TCP_CNx DFB closes the connection and returns to awaiting connection to this local TCP service port.

Note

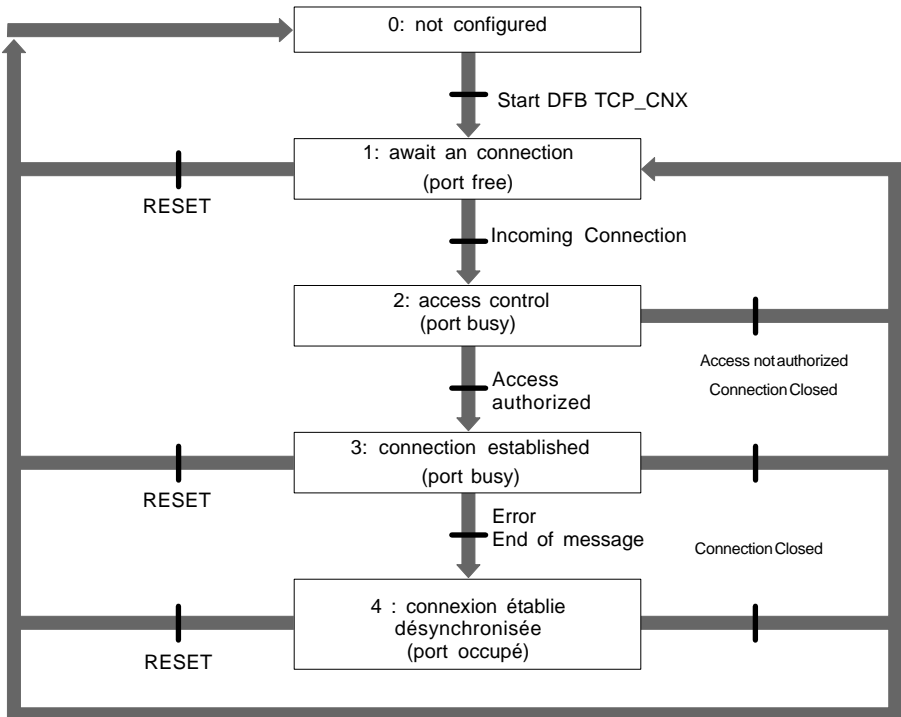
A connection will be closed down immediately by the TCP_CNx DFB for any one of the following reasons:

- Port used by another machine (one connection per port).
- Unauthorized IP address.

Each TCP connection opened with the module is thus characterized individually by the local TCP service port (1 port ÷ 1 connection).

The IP address of the remote machine is an attribute associated solely with this connection

Connection status graph for a port

**Note :**

This status graph, managed by the TCP_CNx DFB is for a TCP service port. The status graph is the same for all the other TCP service ports.

The TCP_CNx must be started for every PLC cycle in order to manage the connections.

Note

An established connection will be closed:

- as a result of a request made by the PL7 application to cancel a transfer in progress.
- upon closure of the connection by the remote application.
- in the event of access by an unauthorized machine (temporary opening for inspection).

Detection of an "end of message" type error upon reception of a message by the module does not result in the connection being automatically closed by the module.

However, the connection will be marked as desynchronized, with the following characteristics:

- All data received on a desynchronized connection is lost.
- New receive requests on a desynchronized connection are immediately sent back (without any data) with a desynchronized connection error.
- It is possible to send data on a desynchronized connection.

All connections will be closed:

- as a result of a software RESET (CPU Reset, downloading of PL7 application).
- as a result of a module equipment RESET (power outage, change of address on terminal block, etc.).
- as a result of a TCP_CNx DFB RESET.

Note

Interrupting connections suddenly without prior warning to the PLC (sudden stoppage of the remote application, power outage of the remote machine, connection closed by the remote machine with cable unplugged, etc.) can give rise to freezing of communication on the port attached to this connection. In practice, a receive request can remain frozen for a maximum of two hours (KEEP ALIVE time-out time). In addition, any new connection to this port during this time period will be refused by the PLC.

In order to remedy this, it is recommended that the PLC application send out a maintain connection message (the message of length 0 will be accepted by the TCP_SEND DFB).

3.2-2 Transferring messages

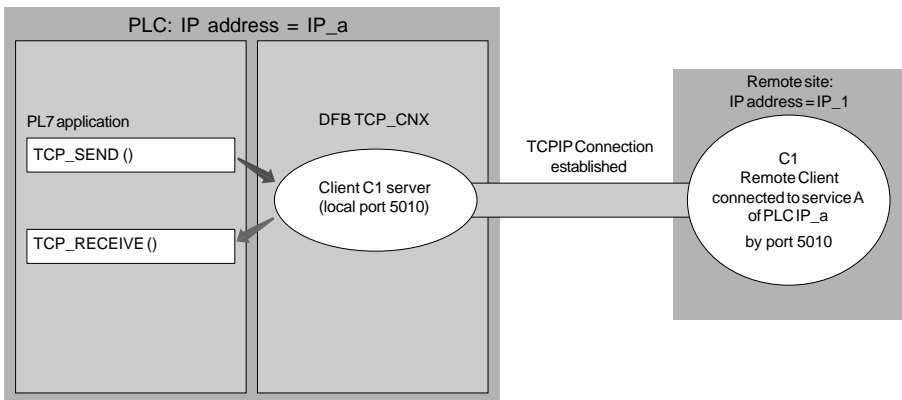
Description of the transfer service

The transfer service is used to exchange **messages** (series of N bytes) on a connection established between a remote client and the PL7 application of the PLC on a given local port.

The transfer requests offer the PL7 application two possible ways of identifying the remote client:

1. By using a single parameter: local port number of the PLC (parameter sufficient for identification of a TCP connection of the module).
The transfer will be made if a connection is established on this port, whatever remote machine is connected to this port.
2. By using the pair of parameters: local port number and IP address of the remote machine.
The transfer will be made if a connection is established on this port with the remote machine specified by the PL7 application.

Transferring a message from application to application



The following message transfer services are available to the PL7 application of the PLC:

- Send request for transmission of a message with a maximum size of 8 kilobytes to a remote client.
A message transmission will remain active as long as the two following conditions are met:
 - the send buffer of the transmitter connection point (PLC) is full,
 - the receive buffer of the destination connection point is full,in other words if the destination is not using the data on the connection.
- Request from a remote client for reception of a message with a maximum size of 8 kilobytes. Await reception is active as long as there is a connection and no complete message has been received on this connection. If necessary, the PL7 application can opt to use a timer (connected to the RST signal of the DFB) to cancel the current request at the end of a given period.

If the TCP connection is not established with the remote client at the time of the data transfer request (send or receive), the request will be refused immediately with the error indication: **connection not established**.

Similarly, if the connection is closed by the remote client or lost, all operations in progress on this connection will be canceled.

Multiple transfer on the same port

The PL7 application cannot activate several requests of the same type (send or receive) on the same port at the same time, without awaiting the report for the previous request.

However, there is no temporal correlation between transmissions and receptions. For one and the same connection, the transmission channel and the reception channel are independent and thus the PL7 application can simultaneously request both a transmission and a reception on the same port.

Format of messages exchanged between the module and the remote application

An important characteristic of communication on a TCP connection is the continuity of the information («byte flows»); the receiving machine cannot identify the structure of a message without the information and the particular protocol established between the sender and the destination.

It has been decided to structure the messages exchanged using the following format:

| 2 bytes | N octets | |
|--|--|-----------------------------|
| Message length = N (network format) | Données du message utilisateur (N-1) octets | Caractère Fin de Message |

At the PLC, the two additional fields (message length and end of message character) will be added (transmission) or deleted (reception) by the DFBs (TCP_SEND or TCP_RECEIVE).

Message length:

- This field specifies the total number of useful bytes in the message, constituted by the user data plus the optional end of message character.
- It is defined as a 16 bit signed value in network format. It is used to encode the values from 0 to 32 767 (32 K - 1).

"End of message" character:

- This 1 byte field is an optional character used to mark the end of the message.
- This end mark becomes redundant when used with the "message length" parameter which is sufficient to structure the flow of data into messages. It is only useful for controlling the coherence of a message when it is being received by the module and for avoiding the spread of an error generated by a remote application.

Procedure for transmission on the TCP connection

The procedure for transmission on an established TCP connection is triggered by the TCP_SEND DFB of the PL7 application.

The TCP_SEND DFB transmission procedure is carried out by block of 240 bytes sent to the TSX ETY 110 WS/5102 module which immediately transmits the block to the remote machine. The "length" and "end of message character" fields (if configured) are automatically transmitted by the TCP_SEND DFB.

The DFB signals the end of the exchange using its activity bit.

Procedure for reception by the module on the TCP connection

The data received on an established TCP connection is stored in the buffers of the TSX ETY 110 WS/5102 module. The data received on the connection will only be read if a TCP_RECEIVE DFB is active.

The size of the receive buffer is a maximum of 4096 bytes. This means that when a message of 8 K is sent by a remote machine, half of the data is stored in its send buffers until the PL7 applications has received the first half of the data.

The TCP_RECEIVE reception procedure is as follows:

- Await/Receive* 2 bytes indicating the length N of the message to be received.
- Receive* N bytes of the message and of the end of message character if the end of message character option is selected in the configuration. Check it and delete it.
- The DFB signals the end of the exchange using its activity bit.

Note

If an "end of message character" error is detected, the receive buffer of the PL7 application contains the received message which can be analyzed.

If there are characters missing relative to the length specified in the message header (sender application error), reception is put on infinite standby whilst awaiting the missing bytes. It will be unlocked upon reception of the following message, but with an "end of message character" error detection (if the end of message character error does not coincide with any useful message data).

Reception is then desynchronized.

An end of message error is not necessarily detected immediately with the first subsequent message (when the end of message character coincides with data in the following messaging, several messages may be received prior to detection).

* The Await/Receive operation on the connection also enables the indication of closure of the connection by the remote machine to be received.

Application to application protocol

No restriction is imposed on the exchange protocol used on a TCP connection established between the remote client and the PLC application. The exchanges can be defined as:

- Request/response at the initiative of the client application.
- Request/response at the initiative of the PLC application.
- Unsolicited messages at the initiative of the client application.
- Unsolicited messages at the initiative of the PLC application.

Characteristics of use of transfer by PL7 application**Number of TCP ports**

The module is dimensioned for 16 TCP ports and thus for 16 connections.

User buffer associated with DFB

During CPU/module exchanges, the send or receive DFB directly uses the buffer specified by the user (no user data is copied into a system buffer). Thus, as long as the DFB is active (ACTIVITY bit at 1), the PL7 program shall under no circumstances modify the contents of the buffer associated with this DFB.

4.1 Points common to all DFBs

The function blocks use the messaging services implemented in the PREMIUM processor. Because of this, **all the communication function blocks can be executed over several PLC cycles.**

Because of the module/CPU exchange architecture, sent messages (8 Kbytes) or received messages (8 Kbytes) will be exchanged between the module and the PL7 application by 256 byte datagrams containing information bytes for checking the segmentation and 240 bytes of useful data.

The CPU/module exchange of a datagram is made per PLC cycle.

4.1-1 Maintained call

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

It is strictly prohibited to execute the same DFB instance more than once in the same PLC cycle .

Maintenance of the DFBs is obligatory since all the communication function blocks are being executed over several PLC cycles.

As soon as the DFB is called, it is restarted for each turn of the PLC cycle as long as its activity bit is equal to 1.

4.1-2 Management table

The PL7 application must supply a 67 word management table to the DFBs in order to enable each DFB to be aware of the status of the connections in progress. Each table entry is constituted by 16 bit words and includes the IP address, the port number and the status of connection with a remote machine.

This table is defined in the PLC in the MW memory zone and according to the following structure:

| | | |
|-----------|-----------|------------------------------------|
| %MWi | MODULE | Module number |
| %MWi + 1 | NP | Number of listen ports |
| %MWi + 2 | EOM | End of message character |
| %MWi + 3 | IP 1 | IP address of remote machine N° 1 |
| %MWi + 5 | P 1 | Number of local port N° 1 |
| %MWi + 6 | Status 1 | Status of connection N° 1 |
| %MWi + 7 | IP 2 | |
| | | |
| %MWi + 62 | Status 15 | |
| %MWi + 63 | IP 16 | IP address of remote machine N° 16 |
| %MWi + 65 | P 16 | Number of local port N° 16 |
| %MWi + 66 | Status 16 | Status of connection N° 16 |

The structure of this table is given for information purposes. This table is entirely managed by the DFBs and must not be modified without activating the RESET input of the connection management DFB.

• Internal data

MODULE – Module number

Number of the physical slot in which module TSX ETY 110 WS/5102 is located in the rack.

NP - Number of listen ports

This parameter, at between 1 and 16 inclusive, defines the NPs, the first elements of the table of the local port numbers to be used.

EOM – End of Message Character

This parameter defines the presence and value of an "End of Message" character to be used in message exchanges. It is updated by the TCP_CNX DFB when it first calls or after a RESET with the value parametered by the PLC application:

- EOM = 0: no end of message character.
- EOM from 0x01 to 0xFF : value of end of message character:
 - to be added during transmission,
 - to be checked and deleted during reception.

IP i – IP address of the client connected to local port No. i

This parameter is updated by the TCP_CNX DFB when a client is connected (value 0 if there is no connection on this port). It is then used by the TCP_SEND and TCP_RECEIVE DFBs to check the "IP address" input parameter when this is not set to zero.

P i – Number of local port No. i

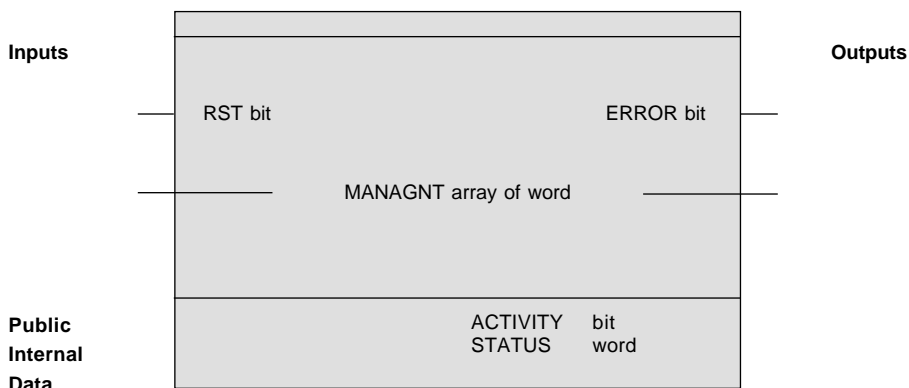
Table of 16 local service ports to be listened to (signed value ≥ 5010).
For a given configuration, only the first number NPs will be used.

Status i – connection status No i

This parameter is used to provide information on the associated connection:

- Bits 0..3: Service socket number of between 0 and 15.
- Bit 8: 1 $\bar{0}$ Established connection 0 $\bar{0}$ no connection.
- Bit 9: 1 $\bar{0}$ Sending.
- Bit 10: 1 $\bar{0}$ Receiving.
- Bit 11: 1 $\bar{0}$ Error on last transmission.
- Bit 12: 1 $\bar{0}$ Error on last reception.

4.1-3 RST and ACTIVITY bits



All the TCP function blocks use the messaging system and are executed over several PLC cycles. This is why the user must be able to interrupt the execution of a block (when awaiting a response to a message); he must also be in a position to know whether or not any blocks are currently being executed, in order to know whether or not to maintain its activity.

ACTIVITY bit

TCP function blocks have a BIT type variable which indicates their activity; in other words whether they have to be maintained or not.

RST action

TCP function blocks have a BIT type RST input which can be used to interrupt their execution when this bit is set to 1, which gives rise to the following actions (if the connection is open):

- Any messages in progress (send and receive) are lost for the connection.
- The connection is closed.

4.1-4 ERROR bit - STATUS WORD

This output bit is set to 1 if the DFB is not able to function correctly. The STATUS word then indicates the type of error.

• STATUS word

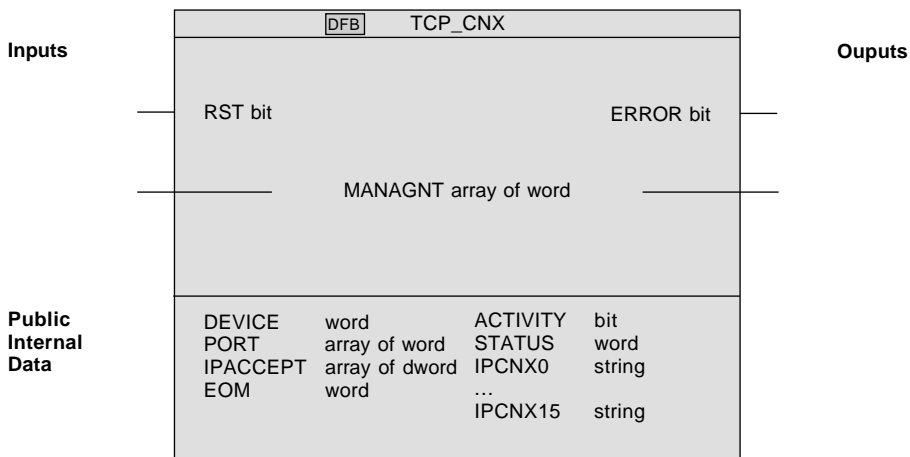
| Bit | DFB | Meaning | Corrective action |
|------------|-------|--|--|
| Bit 0 = 1 | C R S | The module is not an ETY 110 module | Check the value used for the DEVICE input parameter of the configuration. Check the hardware configuration of the PLC. |
| Bit 1 = 1 | C R S | Syntax error | DFB TCP_CNx RESET. Check the parameters and restart. |
| Bit 2 = 1 | R S | The port requested is used by a machine whose IP address is different from that specified in the input parameter (DEST for transmission, FRM for reception). | Check the configuration of the remote clients. Enter the DEST or FRM parameter to select the remote client correctly. |
| Bit 3 = 1 | R S | Local port incorrect. The port number is not in the list of local ports given during module configuration. | Check the list of listen ports of the TCP_CNx DFB. |
| Bit 4 = 1 | R S | Erroneous length | During transmission check that the SIZE parameter is less than 8192. During reception a message header with a field length of > 8192 has been received. |
| Bit 5 = 1 | R S | The connection is not open | Check that there is a client. |
| Bit 6 = 1 | R S | Loss of current connection functioned | Check that the client transfer has correctly. |
| Bit 7 to 9 | | Not used | |

| Bit | DFB | Meaning | Corrective action |
|------------|-----|--|---|
| Bit 10 = 1 | R | Connection desynchronized during reception. A message with an "end of message" character error has been received. | Analyze the message in the receive buffer to identify the source of the error. |
| Bit 11 = 1 | R | End of Message Character error. An "end of message" character error has been detected on this message | Analyze the message in the receive buffer to identify the source of the error. RESET the DFB. |
| Bit 12 = 1 | R | Message truncated. The receive buffer is too small to receive the whole of the message sent by the remote client | Provide a larger buffer. Check the maximum size of the messages exchanged between applications on this connection. |
| Bit 13 = 1 | R | Not used | |
| Bit 14 = 1 | R S | Processing interrupted. The function block has been interrupted during execution by an action RST ou une reprise à restart. | Restart the connection from the client. The server automatically goes into listen mode. |
| Bit 15 = 1 | | Not used | |

5.1 Presentation

The TCP_CNX DFB is responsible for managing the connections with remote clients as well as the operating modes of these connections. It is informed of breaks in connections via the intermediary of the status of the management words by a bit set by a TCP_SEND or TCP_RECEIVE DFB instance.

Only one TCP_CNX DFB instance shall exist for a given TSX ETY 110 WS/5102 module.



Input parameters

| Parameters | Type | Description |
|------------|------|---|
| RST | Bit | Setting this input to 1: - interrupts the exchanges in progress, - causes all connections to close. |

Input/Output parameters

| Parameters | Type | Description |
|------------|------------|--------------------------------------|
| MANAGNT | Word table | Management table common to all DFBs. |

Output parameters

| Parameters | Type | Description |
|------------|------|---|
| ERROR | Bit | BitThis output bit is set to 1 if the DFB is not able to function correctly. The STATUS word indicates the type of error which occurred. |

Internal public data

| Parameters | Type | Write/Read Variables | Description |
|-----------------|------------------|----------------------|--|
| DEVICE | Word | E | Module number Number of the physical slot in which module TSX ETY 110 WS/5102 is located in he rack. |
| PORT | Word table | E | Local port number No. i. Table of 16 local service ports to be listened to (signed value ≥ 5010). The value 0 indicates a non-significant value. |
| IPACCEPT | Double | E | IP address of the remote machine No. i word 4 byte IP address table from table 8 remote machines authorized to connect to the PLC. The value 0 indicates a non-significant value. |
| EOM | Byte | E | "End of Message" character This parameter defines the presence and the value of an "End of Message" character to be used in the message exchanges: - EOM = 0: no end of message character, - EOM from 0x01 to 0xFF: value of end of message character: - added by the module during transmission - checked and deleted by the module during reception. |
| IPCNXi | Character string | L | IP address of the remote machine No. i Read-only character string to be used for maintenance only and indicating the IP address of the remote machine in the format aaa.bbb.ccc.ddd connected to the port with index i in the configuration. The value 0.0.0.0 indicates that no machine is connected to this port. |
| ACTIVITY | Bit | L | This output bit is at one when the DFB is in progress and needs to be maintained. It is set to 0 upon a warm or cold restart and upon a DFB RESET. |
| STATUS | Mot | L | This word is only meaningful if the ERROR output bit (erroneous exchange) is set to 1. It indicates the code of the error which occurred during the exchange (Each word bit set to 1 indicates an error). Please refer to the chapter "Points in common to all DFBs". |

5.1-2 Operation

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

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

If the dialog with the module is not correct, the output error bit ERROR (erroneous exchange) is set to 1.

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

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

5.1-3 Programming example - Connection to 1 single TCP port

• Data used

| | |
|----------------------|---|
| %MW0 : | Management table |
| %KD200 : H'5A000001' | IP address (90.0.0.1) of the remote machine authorized to connect |
| %KW300 : 5010 | Service port |
| %KW0 : 2 | Physical slot in which module ETY110 is located |
| %KW1 : 15 | End of message character |
| %M0 | Error bit |
| %MW100 : | Working variable for indexing of tables |
| TCP_COX | TCP_CNX DFB instance name |

```
! < TRANSMISSION OF MESSAGE >
IF NOT TCP_COX.ACTIVITY THEN

    (* Initializing configuration *)
    TCP_COX.DEVICE := %KW0 ;

    (* prior setting to 0 of table *)
    FOR %MW100 :=0 TO 15 DO
        TCP_COX.PORT[%MW100]:=0 ;
    END_FOR;
    (* List of listen ports *)
    TCP_COX.PORT[0]:= %KW300;

    (* prior setting to 0 of table *)
    FOR %MW100 :=0 TO 7 DO
        TCP_COX.IPACCEPT[%MW100]:=0;
    END_FOR;
    (* List of IP addresses authorized to connect *)
    TCP_COX.IPACCEPT[0]:= %KD200;

    (* End of message character *)
    TCP_COX.EOM:= %KW1;

    (* DFB start *)
    TCP_COX (0, %MW0:67, %M0 );

ELSE

    (* previous message REPORT PROCESSING *)
    IF %M0 THEN JUMP %L3;
    (* Maintenance of DFB *)
    TCP_COX(0, %MW0:67, %M0 );
    END_IF;

END_IF;
```

Note:

Parameters passed on to TCP_CNX DFB

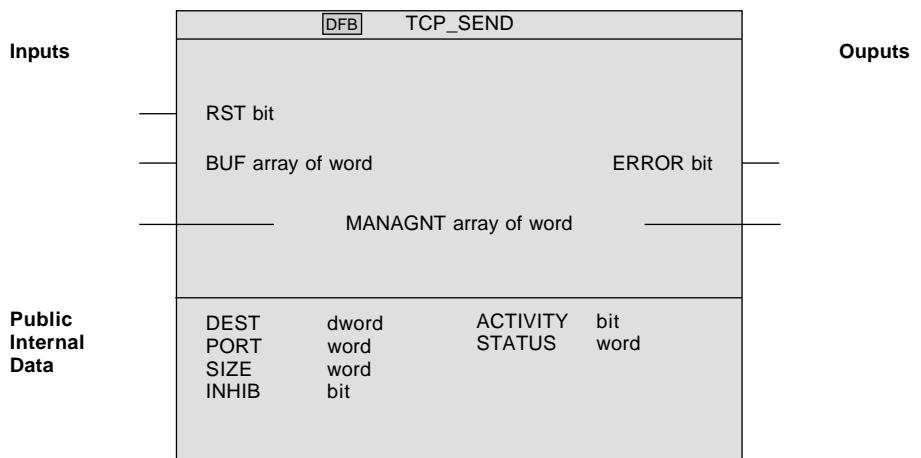
< DFB TCP_CNX instance name > (RST, MANAGNT, ERROR)

6.1 Presentation

B

The TCP_SEND DFB enables a data message to be sent to a remote "client" application via a TCPIP connection. The maximum size of the message to be sent is 8 K bytes.

6.1-1 Characteristics



Input parameters

| Parameters | Type | Description |
|------------|------------|---|
| RST | Bit | Setting this bit to 1: - interrupts the exchange in progress (if the ACTIVITY bit was at 1) - causes the connection to close (if it is open) - sets the ACTIVITY information bit to 0 and the ERROR output bit to 1. The error code is then contained within the word STATUS. |
| BUF | word table | This variable defines the address of the first word %MWi of the message to be sent |

Input/Output parameters

| Parameters | Type | Description |
|----------------|------------|--------------------------------------|
| MANAGNT | Word table | Management table common to all DFBs. |

Output parameters

| Parameters | Type | Description |
|--------------|------|---|
| ERROR | Bit | This output bit is set to 1 if the exchange is not carried out correctly. The STATUS word indicates the type of error that has occurred. |

Internal public data

| Parameters | Type | Write/Read Variable | Description |
|-----------------|-------------|---------------------|--|
| DEST | Double word | E | This variable defines the IP address of the remote machine on which the client is connected to the local port PORT. By default DEST = 0, the message is sent to the only remote client station connected to the local service port of the PLC. |
| PORT | Word | E | This variable defines the local port number to which the remote client is connected. |
| SIZE | Word | E | This variable defines the size in octets from the message to be sent. From 0 to 8192. Does not take account of the end character. |
| ACTIVITY | Bit | L | This bit is at one when an exchange is in progress. It is set to 0 when the exchange is finished. |
| INHIB | Bit | L | This bit enables the error warning to be inhibited: the ERROR output bit and the STATUS word remain at 0 (execution of the block is not interrupted). |
| STATUS | Word | L | This word is only significant if the ERROR output bit (erroneous exchange) is set at 1. It indicates the code of the error that occurred during the exchange (each word bit set to 1 indicates an error). Please refer to the chapter "Points common to all DFBs". |

6.1-2 Operation

The transfer of data is triggered when the TCP_SEND DFB is called if the RST bit is at 0 and if the internal ACTIVITY bit is at 0 (no exchange in progress). During the exchange, the ACTIVITY bit is at 1. At the end of transmission, the ACTIVITY bit is set to 0. In addition, if the exchange is not correct, the ERROR output bit (erroneous exchange) is set to 1.

At any time, setting the RST bit (priority input) to 1 allows the exchange in progress to be interrupted. The ACTIVITY bit (exchange finished) is set to 0 and the ERROR bit (erroneous exchange) is set to 1. The status word <TCP_SEND instance name>.STATUS indicates the type of exchange error. If the connection is open, it closes and the module returns to awaiting the incoming connection. Reestablishing the connection to this TCP service port, reinitiates the transmission of messages. This reconnection is managed by the TCP_CNX DFB.

On triggering of transmission of a message, processing is as follows:

- Checking of the input parameters: IP address of the remote machine, number of local port.
- Search for the open connection with the remote client requested.
- Transmission of the message to be sent per block of 240 bytes (adding the “end of message” character, if the option is requested by default) on the TCP connection.

6.1-3 Programming example

• Data used

| | |
|----------------------|--|
| %MW0: | Management table |
| %KD200 : H'5A000001' | IP address (90.0.0.1) of the remote machine (in 4 bytes) |
| %KW300 : 5010 | Service port |
| %W200 : 400 | Size in words of the message to be sent |
| %W300 to W700 | Message to be sent |
| %M0 | Error bit |
| TCP_EMIS | TCP_SEND DFB instance name |

```
! < TRANSMISSION OF MESSAGE >
IF NOT TCP_EMIS.ACTIVITY THEN
    (* previous message REPORT PROCESSING *)
    IF %M0 THEN JUMP %L3;

    (* Transmission of next message *)
    TCP_EMIS.PORT:=%KW300;
    TCP_EMIS.DEST:=%KD200;
    TCP_EMIS.SIZE:=%MW200*2;
    (* start of DFB first parameter which represents RST at 0 *)
    TCP_EMIS(0, %MW300:400, %MW0:67, %M0);
    END_IF;

ELSE
    (* maintenance of DFB first parameter which represents RST at 0 *)
    TCP_EMIS(0, %MW300:400, %MW0:67, %M0);
    END_IF;
```

Note:

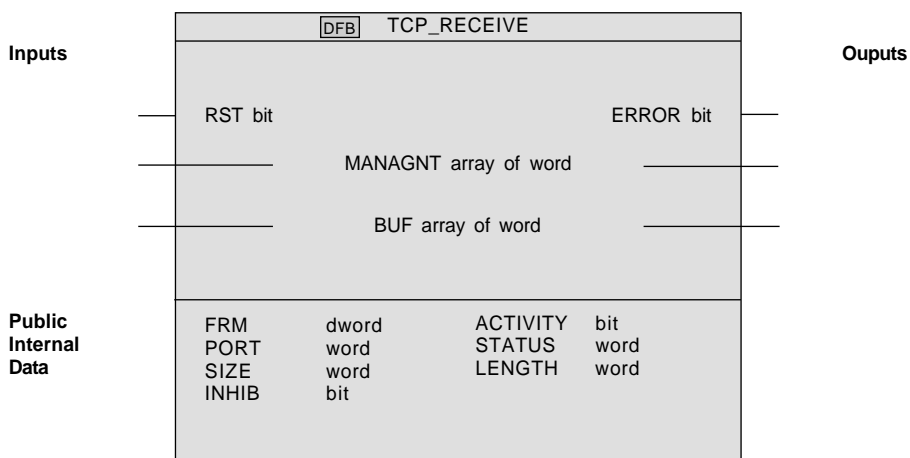
Parameters passed on to DFB

<TCP_SEND> instance name (RST, MANAGNT, BUF, ERROR)

7.1 Presentation

The TCP_DFB enables a data message to be received from a remote "client" application via a TCPIP connection. The maximum size of the message to be sent can be up to 8 K bytes.

7.1-1 Characteristics



Input parameters

| Parameters | Type | Description |
|------------|------|--|
| RST | Bit | Setting this bit to 1: - interrupts the exchange in progress (if the ACTIVITY bit was at 1) - causes the connection to close (if it is open) - sets the ACTIVITY variable to 0 and the ERROR output bit to 1. The error code is then contained within the STATUS word. |

Input/output parameters

| Parameters | Type | Description |
|----------------|------------|---|
| BUF | Word table | This input and output parameter defines the address of the first %MWi word of the buffer in order to receive the message. |
| MANAGNT | Word table | Management table common to all DFBs. |

Output parameters

| Parameters | Type | Description |
|--------------|------|---|
| ERROR | Bit | This output bit is set to 1 if the exchange is not finished correctly. The STATUS word indicates the type of error which occurred. |

internal public data

| Parameters | Type | Write/Read Variable | Description |
|-----------------|-------------|---------------------|--|
| FRM | Double word | E | This variable defines the IP address of the remote machine on which the client is connected to the local port PORT. By default FRM = 0 and the DFB awaits the message sent by the unique remote client connected to the local service port of the PLC. |
| PORT | Word | E | The variable defines the number of the local port to which the remote client is connected. |
| SIZE | Word | E | Cette variable définit la taille en octets du tampon pour recevoir le message. From 0 to 8192. |
| ACTIVITY | Bit | L | This variable is set to 0 by the DFB when the exchange is finished. If the output ERROR bit (erroneous exchange) is at 0, the ACTIVITY variable indicates that the message has been correctly received. However, if the ERROR bit is status 1, the ACTIVITY variable indicates that the exchange is finished but erroneous |
| INHIB | Bit | E | This bit enables the error warning to be inhibited: the ERROR output bit and the STATUS word remain at 0 (execution of the block is not interrupted) |
| STATUS | Word | L | This word is only significant if the ERROR output bit (erroneous exchange) is set at 1. It indicates the code of the error that occurred during the the exchange (each word bit set to 1 indicates an error). Please refer to chapter "Points in common to all DFBs" |
| LENGTH | Word | L | This variable contains the number of bytes received if the ERROR output bit (wrong exchange) is in status 0. |

The transfer of data is triggered when the TCP_RECEIVE DFB is called if the RST input is at 0 and if the ACTIVITY bit is at 0 (no exchange in progress). During the exchange, the ACTIVITY bit is at 1. At the end of reception, the ACTIVITY bit is set to 0. In addition, if the exchange is not correct, the ERROR output bit (erroneous exchange) is set to 1.

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

On triggering the reception of a message, processing is as follows:

- Check the input parameters: identification of the remote machine, number of local port,...
- Search for the open connection with the remote application requested.
- Reception EF queued to await message on this connection.

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

As soon as the full message is copied onto the PLC:

1. The DFB checks the end of character message (if this option has been configured).
2. The STATUS, LENGTH parameters of the DFB are filled in and the ACTIVITY variable and the ERROR bit are then set for the PL7 application.

7.1-3 Programming example

• Data used

| | |
|----------------------|--|
| %MW0 : | Management table |
| %KD200 : H'5A000001' | IP address (90.0.0.1) of the remote machine (in 4 bytes) |
| %KW300 : 5010 | Service port |
| %W200 : 400 | Size of the receive buffer in words |
| %W300 à W700 | Receive buffer |
| %M0 | Error bit |
| TCP_RECEP | DFB TCP_RECEIVE instance name |

```
! < TRANSMISSION OF MESSAGE >
IF NOT TCP_RECEP.ACTIVITY THEN
    (* previous message REPORT PROCESSING *)
    IF %M0 THEN JUMP %L3;

    (* Transmission of next message *)
    TCP_RECEP.PORT:=%KW300;
    TCP_RECEP.FRM:=%KD200;
    TCP_RECEP.SIZE:=%MW200*2;
    (* DFB start*)
    TCP_RECEP(0, %MW0:67, %MW300:400, %M0);
    END_IF;
ELSE
    (* maintenance of DFB first parameter which represents RST at 0 *)
    TCP_RECEP(0, %MW0:67, %MW300:400, %M0);
    END_IF;
```

Note :

Parameters passed on to DFB

<TCP_RECEIVE> instance name (RST, MANAGNT, BUF, ERROR)

8.1 Points relevant to the performance of a TCPIP-OPEN communication

The performances of a TCPIP-OPEN communication depend on the following points:

- The PLC cycle time

This takes account of the process handling time and the processing time reserved for the TCPIP-OPEN communication.

- The periodic cycle time

This shall take into account the processing time of the TSX ETY 110 WS/5102 module in order for the response to the request made to the module to arrive during the following PLC cycle.

- The processing time of the TSX ETY 110 WS/5102 module. This is less than 30 ms for 4 configured TCP ports.
- The maximum number of EFs started during the same PLC cycle. This is linked to the type of processor.

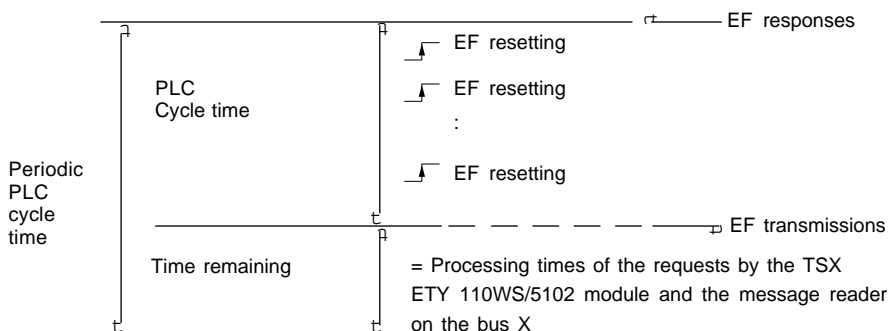
Note

The best performances are obtained with a regular communication transmission and reception flow. This operation is optimal when a DFB can send an EF during each PLC cycle.

As the EF is sent at the end of the PLC cycle, it is essential that its response arrives at the start of the following period.

For this, the read time of the message on the X bus and the processing time of the ETY110WS/5102 module shall be less than the time remaining between the end of the PLC cycle and the start of the following period.

8.2 Diagram of EF processing during a PLC cycle



Note:

The TCPIP-OPEN EFs are used by the TCP_CNXX, TCP_SEND, TCP_RECEIVE DFBs. Please refer to the document "OPEN TCP for TSX Premium" relating to the structure and operation of the TCPIP-OPEN EFs.

8.3 Performance measurements

The performance measurements were made using a TCPIP-OPEN server PLC application on Premium TSXP57352 and a client TCPIP-OPEN application on a PC running Windows NT.

| Periodic cycle time of a PLC | 50 ms | 70 ms | 80 ms | 90 ms |
|--|-------|--------|-------|--------|
| Response time for a 8 K octets message sent to 4 TCP service ports | 3,7 s | 3,33 s | 3 s | 3,75 s |
| Response time for a 8 K octets message sent to 2 TCP service ports | 3 s | | | |

9.1 Differences with OFB PL7-3

The offer is similar on the Series 7 range with ETH110. The differences are mainly:

- **Performance:** The ETH110 module only supports the TCPIP_OPEN communication profile. The performances with the TSX ETY 110 WS/5102 module will depend on the use of the other module functions (UNITE messaging system, MODBUS, HTTP, SNMP, FTP messaging system).

The execution of several **simultaneous** DFBs (during the same PLC cycle) for making exchanges of the same nature with the same remote machine (identified by the port number and/or IP address) is not possible as it cannot be managed in the TCOPEN architecture.

- **Operating modes:** The connections are managed by the module in the ETH110 offer whereas it is managed by the PLC application in the current approach with TCOPEN.
- **Configuration:** On ETH110, the configuration consists of the module parameters, the list of ports used as well as the list of authorized remote IP addresses. In the TCOPEN offer, the module parameters are configured by PL7 and managed by the system. The list of ports used and authorized IP addresses is managed by the TCP_CNX DFB (or EF).
- **PL7 programming:** The programming method is different: The DFBs must be maintained by the application, whereas the OFBs of the ETH110 are maintained automatically.

The **Echo function does not** exist in the TCOPEN architecture.

- **Network and remote external view:** The message entity is recognized by the ETH110 module whereas only the PLC recognizes it in the TCOPEN offer. When sending a message this involves the **fragmentation into 240-byte units on the network. Transmission interruption** by the Premium of a message in progress may lead to the end of the message being lost.
When receiving a message, the network view is identical to the behavior of the ETH110.
- **Diagnosis:** The diagnostic tools used for the OFBs in the 7 series (Applidiag) must be replaced by standard DFB operating tools built into the PL7.

9.2 Reminder on the TCPIP communication

Please refer to the following documents:

- Préparation au MCSE TCPIP, S & SM publications.
- Préparation au MCSE TCPIP, CAMPUSPRESS publications.
- TCPIP by Douglas Comer, InterEditions publications.