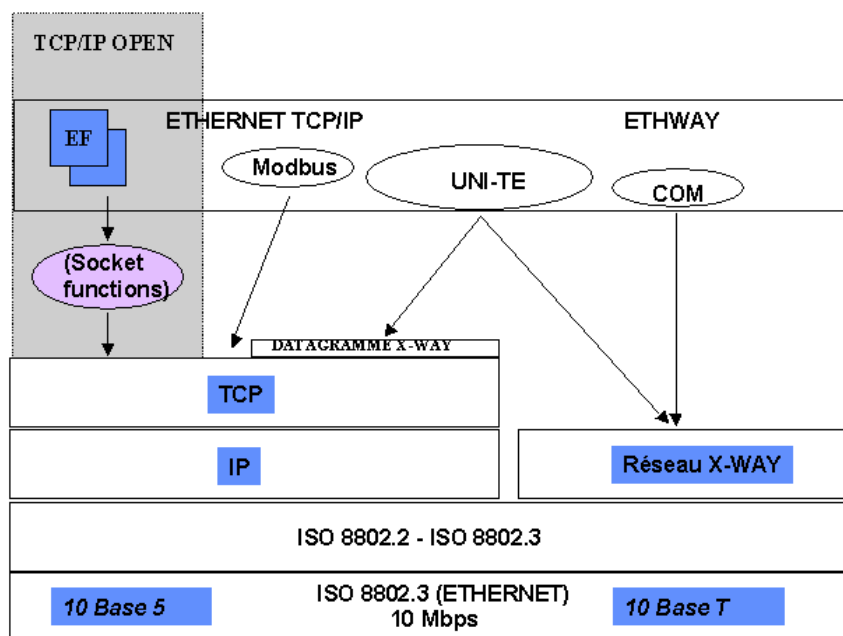


Chapitre	Page
1 Présentation	1/1
1.1 Introduction	1/1
2 Installation	2/1
2.1 Installation de la bibliothèque TCP/IP	2/1
2.2 Installation des EF du kit OPEN TCP	2/1
2.2-1 Installation des fichiers exécutables	2/1
2.2-2 Installation des fichiers sources	2/1
2.3 Introduction	2/2
2.4 Vue d'ensemble de l'utilisation et de la gestion des sockets	2/3
2.5 Structure générale d'une fonction de communication par TCP/IP	2/4
2.6 Fonction socket()	2/8
2.7 Fonction bind()	2/10
2.8 Fonction listen()	2/11
2.9 Fonction accept()	2/12
2.10 Fonction setsockopt()	2/14
2.11 Fonction select()	2/16
2.12 Fonction send()	2/18
2.13 Fonction recv()	2/20

Chapitre	Page
2.14 Fonction shutdown()	2/22
2.15 Fonction close()	2/24
3 Taille du code	3/1
4 Modes de fonctionnement	4/1
5 Diagnostic	5/1
6 Performances	6/1
6.1 Nombre de connexions	6/1
6.2 Echange de données	6/1
7 Notes relatives aux applications	7/1
7.1 Fonctions élémentaires de niveau 1 pour programmation avancée	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_STSKOPT_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 Modèle client/serveur	7/22
7.3 Client/serveur : modes de fonctionnement	7/23
8 Glossaire	8/1

1.1 INTRODUCTION

OPEN TCP pour PREMIUM est un ensemble de composants qui permet de créer des fonctions élémentaires (EF, Elementary Functions) pour les communications par TCP/IP d'une application Premium.



Descriptif du produit OPEN TCP :

OPEN-TCP est basé sur une nouvelle évolution (V2.8 et ultérieure) du module ETY5102. Il est livré sur un CD-ROM séparé. Après l'installation, le répertoire PL7 contient les répertoires suivants :

- Répertoire PL7\OFLIB32\LibTcp\ : la bibliothèque TCP/IP à ajouter à l'utilitaire PL7 SDKC. Elle est basée sur un simple sous-ensemble des API pour sockets de Berkeley, et contient des fonctions permettant de créer un serveur de connexion TCP/IP et d'échanger des données.
- Répertoire PL7\OFLIB32\ : les spécifications techniques (ce document)
- Répertoire EFL1-adv\ : autre sous-ensemble (fichiers exécutables) des fonctions élémentaires de niveau 1, avec une interface spécifique adaptée à la programmation avancée.
- Répertoire PL7\OFLIB32 : un exemple de modèle de serveur Telnet avec une application PL7.
- Répertoire PL7\OFLIB32\Dvt_ef\ : contient les sous-répertoires suivants :
- Répertoire PL7\OFLIB32\Fam_fffc\ : fichiers source des fonctions élémentaires pour programmation avancée

Les outils de développement suivants sont nécessaires, mais ne sont pas livrés :

- Environnement PL7 Pro, version 3.3 IE 72 ou ultérieure.

Les outils de développement suivants sont nécessaires pour développer de nouvelles fonctions élémentaires, mais ne sont pas livrés :

- PL7 SDKC V3.3 IE 18 ou ultérieure, pour permettre aux développeurs en C de créer des fonctions élémentaires (EF) dans l'environnement PL7.

Version requise de Premium :

- V3.3 ou ultérieure

REMARQUE

Pour utiliser la bibliothèque, il est nécessaire de disposer d'une connaissance minimale de TCP ainsi que des sockets, et de respecter les règles d'implémentation des EF à l'aide du SDK C. Le client et le serveur nécessitent un ensemble de conventions courantes pour que le service puisse être effectué et accepté. Cet ensemble de conventions se compose d'un protocole qui doit être implémenté de part et d'autre de la connexion.

L'utilisateur est responsable du développement des EF à l'aide des utilitaires fournis et de la gestion de leurs modes de fonctionnement.

Des profils privés (services TCP/IP, profil ETHWAY, échanges CWE, SNMP) sont disponibles, mais leurs performances peuvent varier selon l'utilisation qui est faite de la bibliothèque OPEN TCP.

La mise en œuvre et la maintenance d'OPEN TCP sont réservées à du personnel ayant une connaissance qualifiée des sockets, et ce document ne doit pas être considéré comme représentant une formation suffisante pour toute personne qui ne serait pas, par ailleurs, qualifiée pour développer à l'aide de ce SDK. Bien que toutes les précautions raisonnables aient été prises pour fournir ici des informations exactes et dignes de foi, Schneider Automation ne saurait en aucune manière être tenu pour responsable des conséquences de l'utilisation de ce document.

Limites de responsabilité de Schneider Automation :

Schneider Automation ne saurait être tenu pour responsable des actes suivants :

- La conception et la validation de l'architecture du système de communication (modes de fonctionnement et protocoles client/serveur, performances, ..)
- L'implémentation d'EF appropriées (ou de l'utilisation des exemples d'EF inclus dans le kit OPEN TCP)
- Les tests et la validation des EF intégrées à l'architecture du système de communication
- La maintenance et les erreurs de diagnostics

Schneider Automation ne saurait en aucun cas être tenu pour responsable de toute perte, de tout dommage ou de tout accident bénin ou mortel résultant d'une modification ou d'une utilisation erronée des produits par le Client.

2.1 Installation de la bibliothèque TCP/IP

Pour créer vos propres fonctions élémentaires dans le SDKC pour PL7 à l'aide de la bibliothèque TCP/IP, vous devez d'abord installer celle-ci.

Pour installer la bibliothèque TCP/IP, il est nécessaire d'avoir installé préalablement le SDKC pour PL7 sur le système de développement.

Pour installer la bibliothèque, copiez les fichiers "option.txt" et "TCP/IP.lib" dans le répertoire Sdkcefxy/lib, écrasez le fichier "syssdkc3.lib" dans Sdkcefxy/lib et copiez le fichier "TCP/IP.h" dans le répertoire Sdkcefxy/inc. Dans ces deux noms de répertoires, xy représente le numéro de version du SDKC pour PL7 ; cette version doit être au minimum la version 3.3 IE 18.

Le SDKC pour PL7 est maintenant prêt à accepter la programmation pour TCP/IP.

2.2 Installation des EF du kit OPEN TCP

2.2.1 Installation des fichiers exécutables

Pour utiliser les fonctions élémentaires du produit Open TCP avec l'outil de programmation PL7, vous devez installer les fichiers exécutables :

Pour ce faire, l'outil SDKC pour PL7 n'est pas nécessaire.

Fonctions élémentaires de niveau 1 pour programmation avancée

Insérez le CD-ROM dans le lecteur du PC et sélectionnez le programme d'installation des DFB et des EF.

2.2.2 Installation des fichiers sources

Pour visualiser ou modifier les fonctions élémentaires du produit Open TCP avec l'outil SDKC pour PL7, vous devez installer les fichiers sources à la suite de l'installation de la bibliothèque TCP/IP.

Pour installer les fichiers sources, il est nécessaire d'avoir installé préalablement le SDKC pour PL7 sur le système de développement.

Le répertoire suivant doit être copié dans le répertoire Dvt_ef\ du répertoire d'ensemble "EF Family" de l'outil SDKC pour PL7 :

- Répertoire Fam_fffc\ : fichiers source des fonctions élémentaires pour programmation avancée

2.3 Introduction

Toutes les fonctions de communication par TCP/IP sont exécutées dans un contexte d'EF ; il ne s'agit donc pas de fonctions de blocage. Chaque appel à une fonction de la bibliothèque déclenche une transaction avec le module Ethernet ETY5102, qui est le véritable exécuteur du service. La transaction débute à la fin du cycle PLC. Le développeur doit donc tester le bit d'activité qui indique la fin de la fonction.

Le nombre des fonctions de socket fournies et le nombre de possibilités de ces fonctions sont volontairement réduits. Certains paramètres et options d'utilisation des sockets sont imposés par le module ETY5102. Voir la description de la fonction **socket()**.

Il est possible d'appeler plusieurs services TCP/IP dans le même cycle PLC, mais il n'est pas certain que les traitements seront exécutés dans l'ordre chronologique des appels. **Il est donc important d'attendre l'exécution d'une fonction avant de demander un nouveau service sur le même socket** (ex : attendre le retour de **socket()** avant d'appeler **bind()**, attendre le retour de **bind()** avant d'appeler **listen()**, etc.). Pour les échanges de données (fonctions **send()** et **recv()**), l'utilisateur est responsable de l'ordre des messages.

Règles d'implémentation des EF :

Il n'est pas possible de créer plus d'une EF par classe. Le respect de cette règle est indispensable pour que le système de PLC puisse traiter l'opération.

Tout ajout d'une nouvelle EF sans changer le nombre de classes dans le SDK C provoquera une défaillance du système et une destruction de l'application.

2.4 Vue d'ensemble de l'utilisation et de la gestion des sockets

Un socket est un point d'extrémité de communication. Il représente le module constitutif d'une communication. Les communications s'effectuent par envoi et réception de données via des sockets. La bibliothèque TCP/IP ne comporte que des sockets de flux utilisant TCP et prenant en charge un service de communication à base de connexions.

Dans l'architecture Premium, les descripteurs de sockets sont des numéros de 1 à 32.

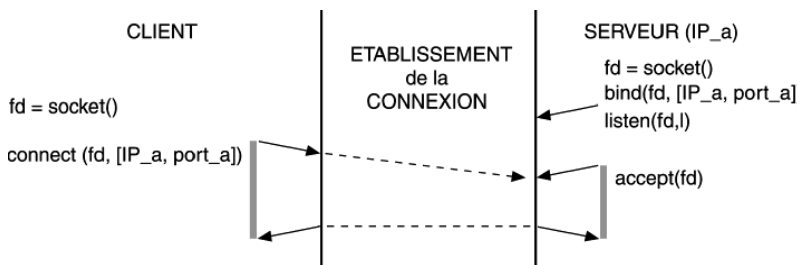
Les numéros 1 à 16 sont affectés aux sockets créés par la fonction de socket. Ce sont les sockets d'entrée. Les numéros 17 à 32 sont affectés aux sockets créés par la fonction d'acceptation. Ce sont les sockets connectés.

Les sockets sont créés via la fonction **socket()**. Un numéro de socket est renvoyé par cette fonction ; ce numéro est utilisé par le créateur pour accéder à ce socket.

Les sockets sont créés sans adresses (adresses IP et numéro de port). Pour qu'un socket puisse recevoir des données, il doit avoir été relié à un port. La fonction **bind()** permet de relier un numéro de port à un socket. **bind()** crée une association entre le socket et le numéro de port spécifié. L'adresse IP est directement reliée par l'ETY5102 à son adresse IP locale.

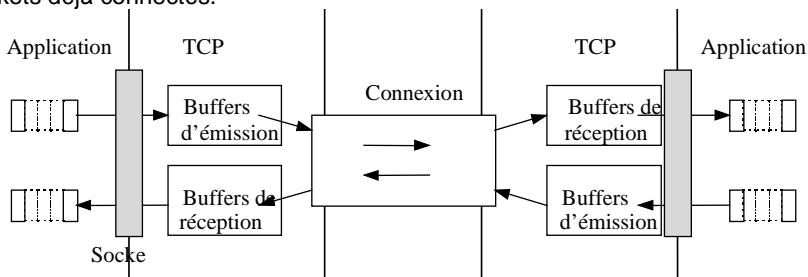
Le PLC est toujours serveur (pour les connexions TCP) ; il doit donc relier un port à son socket, puis utiliser la fonction **listen()** pour établir le socket. Celui-ci peut alors accepter les demandes de connexion des clients.

Pour achever l'établissement d'une connexion, le serveur doit émettre la fonction **accept()** en indiquant le numéro de socket qui a été spécifié dans l'appel précédent à **listen()**. Un nouveau socket est créé avec les propriétés du socket initial. Ce nouveau socket est connecté au socket du client, et son numéro est renvoyé au serveur. Le socket initial est alors libre pour les autres clients susceptibles de se connecter au serveur.



Établissement d'une connexion

Après l'établissement d'une connexion, les données peuvent être transférées. Les fonctions **send()** et **recv()** sont spécifiquement conçues pour être utilisées avec les sockets déjà connectés.



Échanges de données sur une connexion TCP

La fonction **setsockopt()** permet au créateur d'un socket d'associer des options à ce dernier. Ces options modifient le comportement du socket.

La fonction **select()** permet au développeur de tester les événements sur tous les sockets.

La fonction **shutdown()** permet à l'utilisateur d'un socket de désactiver la transmission et/ou la réception sur ce socket.

Lorsqu'un socket n'est plus nécessaire, son descripteur de socket peut être abandonné à l'aide de la fonction **close()**.

2.5 Structure générale d'une fonction de communication par TCP/IP

Le traitement des fonctions de communication par TCP/IP est asynchrone par rapport au traitement de la tâche d'application qui les a activées. Une fonction est dite asynchrone lorsqu'elle est exécutée au cours d'une ou plusieurs tâches de PLC consécutives à la tâche qui l'a activée.

Une fonction de communication par TCP/IP utilise :

- un numéro d'interface ;
- des paramètres spécifiques à la fonction ;
- des paramètres de gestion.

Chaque fonction est structurée ainsi :

Fonction (numéro d'interface, paramètres spécifiques, paramètres de gestion)

Numéro d'interface :

Ce paramètre est la position physique du module ETY5102 dans le rack principal (seul le rack numéro 0 est disponible).

Paramètres spécifiques :

Ces paramètres sont spécifiques à chaque type de fonction de communication TCP/IP. Ils sont décrits dans les sections consacrées à chacune de ces fonctions.

Paramètres de gestion :

Les paramètres de gestion sont communs à toutes les fonctions de communication TCP/IP. Ces paramètres sont les suivants :

- un paramètre qui fournit des informations sur l'activité de la fonction ;
- un paramètre qui contient le rapport des échanges (rapport système et rapport de fonction) ;
- un paramètre de longueur indiquant le nombre d'octets à envoyer (fonction **send()**) ou à recevoir (fonction **recv()**),

Ces paramètres nécessitent un tableau de quatre mots consécutifs, dont la structure est la suivante :

	Mot n°	octet 1	octet 0
Données gérées par le système	1	Réservé pour le système	bit 0 : bit d'activité
	2	Rapport de fonction	Rapport système
	3	Réservé pour le système	
Données gérées par l'utilisateur	4	Longueur	

Bit d'activité :

Ce bit signale l'état d'exécution de la fonction. Il est mis à 1 au début de l'exécution, et revient à 0 à la fin.

Rapport système :

Le rapport système représente le résultat des échanges entre l'application et le module ETY5102. Il ne concerne pas l'accès à la pile TCP/IP.

Ce rapport est commun à toutes les fonctions. Il est signifiant lorsque la valeur du bit d'activité passe de 1 à 0. Les diverses valeurs de ce rapport sont indiquées dans le tableau ci-dessous :

Valeur	Rapport système :
16#00	Échange correct
16#01	Échange interrompu à l'expiration du délai (time-out)
16#0B	Pas de ressources système (CPU)
16#07	Module réseau absent (probablement en raison d'un numéro d'interface incorrect)
16#FF	Erreur de communication avec l'ETY5102

Rapport de fonction :

Cet octet décrit le résultat de l'interaction sur la pile TCP/IP du module ETY5102. Il n'est signifiant que lorsque le rapport système contient la valeur 16#00 ou 16#FF.

Si le rapport système vaut 16#00, le rapport de fonction est spécifique à chaque fonction. Il est décrit dans les sections consacrées à chacune de ces fonctions.

Si le rapport système vaut 16#FF (message refusé), les diverses valeurs de ce rapport de fonction sont indiquées dans le tableau ci-dessous :

Valeur	Rapport de fonction
16#0B	Manque de ressources système (trop grand nombre d'EF dans le même cycle de PLC)
16#0C	L'ETY5102 n'est pas en marche

Longueur :

Le paramètre de longueur n'a d'importance pour l'utilisateur qu'avec les fonctions **send()** et **recv()**.

Ce champ peut être modifié par le système dans toutes les fonctions de communication TCP/IP. C'est pourquoi le développeur doit vérifier ce paramètre avant tout nouvel appel à la fonction **send()** ou **recv()**, même s'il est identique à celui de l'appel précédent.

Important :

Lorsque la fonction est en cours de traitement (bit d'activité = 1), les paramètres de gestion ne peuvent pas être modifiés par la tâche d'application.

En raison du caractère asynchrone de ce mécanisme, ces mots doivent être prélevés dans un espace mémoire statique (ex. : MW space).

Il est recommandé de toujours tester les rapports de fonction dès que les fonctions ont été exécutées et avant de les activer à nouveau.

À la première exécution, il est essentiel de vérifier que tous les paramètres de gestion sont remis à 0.

Définition de la structure de gestion

```
typedef struct
{ unsigned char Activity ;
  unsigned char réservé1 ;
  unsigned char SystemReport ;
  unsigned char FunctionReport ;
  unsigned short réservé2 ;
  unsigned shortLength ;
} Mngt ;
```

Définition de socket

```
unsigned short Sock ;
```

2.6 Fonction `socket()`

Syntaxe

```
#include <TCP/IP.h>
```

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

Description

La fonction `socket()` crée un nouveau socket et renvoie son numéro. Le socket est un point d'extrémité de communication TCP/IP. Il est créé sous forme de socket de STREAM TCP avec les options suivantes :

SO_LINGER sans time out	L'option SO_LINGER contrôle l'action à effectuer lorsque les données non transmises sont mises en file d'attente sur un socket et que la fonction close() est appelée. Voir dans la définition de close() la description de l'effet de SO_LINGER sur la sémantique de close() .
NO_DELAY	Désactive l'algorithme d'accusé de réception de délai. Les données sont envoyées immédiatement sur le réseau au lieu d'attendre que la fenêtre soit remplie.
KEEP_ALIVE	Assure la pérennité de la connexion en transmettant à intervalles réguliers par le socket un paquet qui ne fait pas partie du flux de données TCP de l'utilisateur.
REUSEADDR	Le port local peut être réutilisé dans le cadre d'un appel à <code>bind()</code> .

Paramètres reçus

Interface	Spécifie le numéro du module ETY5102
Sock	Pointeur sur un tableau d'un mot pour le numéro du socket.
Mngt	Pointeur sur un tableau de quatre mots pour la gestion

Valeurs renvoyées

Mngt.SystemReport	signale une erreur si diffère de 0 (voir codes d'erreur au chap. 0)
Mngt.FunctionReport	NO_ERROR (0) ENOBUFFS (0x37) : Le nombre maximal de sockets est atteint.
Sock	En l'absence d'erreur, le numéro de socket.

2.7 Fonction bind()

Syntaxe

```
#include <TCP/IP.h>
```

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

Description

Cette fonction permet d'attribuer (ou relier) à un socket une adresse Internet et un numéro de port. Les sockets sont créés sans recevoir d'adresse et ne peuvent donc pas être utilisés pour recevoir des données (hors demande de connexion) tant qu'une adresse ne leur a pas été attribuée.

L'adresse Internet est fixée par l'ETY5102 à sa propre adresse IP locale telle qu'elle a été configurée.

Certains numéros de port sont interdits à l'utilisateur car ils sont déjà utilisés par l'ETY5102. Ces numéros de port sont 20 et 21 (ports FTP), 23 (port Telnet), 67 et 68 (ports DHCP BOOTP), 80 (port HTTP), 161 et 162 (ports SNMP), 502 (port Schneider Automation), 5000 et 5001 (ports spécifiques de l'ETY5102), 1024... (ports TCP USER), 3124... (ports du scanner d'E/S), 7400-8400 (ports RTPS).

Paramètres reçus

Interface	Spécifie le numéro du module ETY5102
SocketNumber	Numéro du socket
PortNumber	Numéro du port à affecter au socket
Mngt	Pointeur sur un tableau de quatre mots pour la gestion

Valeurs renvoyées

Mngt.SystemReport	signale une erreur si diffère de 0 (voir codes d'erreur au chap. 0)
Mngt.FunctionReport	NO_ERROR (0) EBADS (0x09) : Le numéro de socket n'est pas valide EADDRINUSE (0x30) : Le port spécifié est déjà utilisé EADDRNOTAVAIL (0x37) : Le numéro de port spécifié n'est pas disponible EINVALID (0x16) : Le socket fait déjà l'objet d'une liaison

2.8 Fonction listen()

Syntaxe

```
#include <TCP/IP.h>
```

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

Description

Cette fonction configure le socket spécifié pour recevoir des connexions. Les demandes de connexion sont mises en file d'attente sur le socket jusqu'à ce qu'elles soient acceptées à l'aide d'un appel à accept(). La longueur de la file d'attente est fixée à 16. Si une demande de connexion arrive lorsque la file est pleine, le client qui est à l'origine de cette demande reçoit une erreur ECONNREFUSED.

Paramètres reçus

Interface	Spécifie le numéro du module ETY5102
SocketNumber	Numéro du socket
Mngt	Pointeur sur un tableau de quatre mots pour la gestion

Valeurs renvoyées

Mngt.SystemReport	signale une erreur si diffère de 0 (voir codes d'erreur au chap. 0)
Mngt.FunctionReport	NO_ERROR (0) EBADS (0x09) : Le numéro de socket n'est pas valide

2.9 Fonction accept()

Syntaxe

```
#include <TCP/IP.h>
```

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

Description

Cette fonction permet d'accepter une demande de connexion reçue par le socket spécifié depuis un socket extérieur.

Avant l'appel à `accept()`, il est nécessaire d'appeler `listen()` afin de configurer le socket pour qu'il puisse recevoir une demande de connexion. La fonction `accept()` extrait la première demande de connexion dans la file des connexions en attente, crée un socket connecté ayant les mêmes propriétés que le socket original, effectue la connexion entre le socket extérieur et le nouveau socket, puis retourne un numéro pour ce dernier. Le nouveau numéro de socket ainsi renvoyé est utilisé pour transmettre des données au socket extérieur et en recevoir. Il n'est pas utilisé pour accepter d'autres connexions, mais le socket original reste ouvert pour accepter d'autres connexions.

S'il n'y a pas de connexions en attente dans la file, `accept()` renvoie une erreur.

La fonction `accept()` enregistre le numéro du socket connecté dans le tableau `SocketNumber`.

La fonction `accept()` enregistre l'adresse de client du socket connecté dans le tableau `ClientAddr`.

Paramètres reçus

Interface	Spécifie le numéro du module ETY5102
SocketNumber	Numéro du socket
ClientAddr	Pointeur sur un tableau de quatre mots pour stocker l'adresse du client
Mngt	Pointeur sur un tableau de quatre mots pour la gestion

Valeurs renvoyées

Mngt.SystemReport	signale une erreur si diffère de 0 (voir codes d'erreur au chap. 0)
Mngt.FunctionReport	NO_ERROR (0) EBADS (0x09) : Le numéro de socket n'est pas valide EWOULDBLOCK (0x23) : pas de demande de connexion EINVAL (0x16) : La fonction listen() doit avoir été appelée avant accept()
ClientAddr[0]	En l'absence d'erreur, le numéro de socket connecté. (mot)
ClientAddr[1]	En l'absence d'erreur, le numéro du port client. (mot)
ClientAddr[2]	En l'absence d'erreur, le mot de poids faible de l'adresse IP du client
	ClientAddr[3] En l'absence d'erreur, le mot de poids fort de l'adresse IP du client

2.10 Fonction setsockopt()

Syntaxe

#include <TCP/IP.h>

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

Description

La fonction setsockopt() configure les options associées au socket spécifié. Le nombre d'options est volontairement réduit en raison du mode de fonctionnement de l'ETY5102. Certaines options sont définies lors de la création du socket. Voir la description de la fonction **socket()**.

Paramètres reçus

Interface	Spécifie le numéro du module ETY5102
SocketNumber	Numéro du socket
Option	Spécifie l'option à configurer ou à réinitialiser : DONT_ROUTE : Indique que les données transmises ne doivent pas être routées. Les paquets destinés à des nœuds non connectés sont supprimés RESET_DONT_ROUTE : réinitialise DONT_ROUTE KEEP_ALIVE : Préserve la connexion en transmettant un paquet à intervalles réguliers sur le socket RESET_ KEEP_ALIVE : réinitialise KEEP_ALIVE .
Mngt	Pointeur sur un tableau de quatre mots pour la gestion

Valeurs renvoyées

Mngt.SystemReport	signale une erreur si diffère de 0 (voir codes d'erreur au chap. 0)
Mngt.FunctionReport	NO_ERROR (0) EBADS (0x09) : Le numéro de socket n'est pas valide EINVAL (0x16) : Option non valide
Mngt.Length	En l'absence d'erreur, nombre d'octets stockés dans le tampon.

Option	Valeur
DONT_ROUTE	1
RESET_DONT_ROUTE	2
KEEP_ALIVE	3
RESET_KEEP_ALIVE	4

2.11 Fonction select()

Syntaxe

```
#include <TCP/IP.h>
```

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

Description

La fonction select() permet de multiplexer des demandes d'E/S sur plusieurs sockets. Un masque de bits de deux mots est renvoyé par la fonction select() ; il indique les sockets qui ont des événements en attente de traitement.

Dans l'architecture Premium, les descripteurs de sockets sont des numéros de 1 à 32.

Les numéros 1 à 16 sont affectés aux sockets créés par la fonction de socket. Ce sont les sockets d'entrée. Les numéros 17 à 32 sont affectés aux sockets créés par la fonction d'acceptation. Ce sont les sockets connectés. Le premier mot du masque est donc un tableau de bits des sockets d'entrée (le bit 0 correspond au socket 0) et le second mot est un tableau de bits des sockets connectés.

Paramètres reçus

Interface	Spécifie le numéro du module ETY5102
pMask	Pointeur sur un tableau de deux mots pour renvoyer le masque des sockets.
Mngt	Pointeur sur un tableau de quatre mots pour la gestion

Valeurs renvoyées

Mngt.SystemReport	signale une erreur si diffère de 0 (voir codes d'erreur au chap. 0)
Mngt.FunctionReport	NO_ERROR (0)
pMask	Pointeur sur un tableau de deux mots pour renvoyer le masque des sockets. Chaque bit mis à 1 indique un événement sur le socket qui correspond à ce bit. Exemple : si le bit 3 du second mot est mis à 1, le socket numéro 20 doit être lu par la fonction <code>recv()</code> . si le bit 5 du premier mot est mis à 1, le socket numéro 6 doit être lu par la fonction <code>accept()</code> .

Si le bit est mis sur un socket d'entrée, cela signifie que :

- une demande de connexion est en attente.

Si le bit est mis sur un socket connecté, cela signifie que :

- des données sont en attente sur le socket.
- la connexion est interrompue.

2.12 Fonction send()

Syntaxe

```
#include <TCP/IP.h>

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

Description

La fonction send() permet d'envoyer des données à un socket externe. La longueur maximale des données à transmettre est de 240 octets

Il n'est pas possible de transmettre des données hors bande.

Paramètres reçus

Interface	Spécifie le numéro du module ETY5102
SocketNumber	Numéro du socket
pBuf	Pointe sur un tampon contenant les données à transmettre
Mngt	Pointeur sur un tableau de quatre mots pour la gestion
Mngt.Length	Nombre d'octets à transmettre. La longueur maximale est de 240 octets

Valeurs renvoyées

Mngt.SystemReport	signale une erreur si diffère de 0 (voir codes d'erreur au chap. 0)
Mngt.FunctionReport	NO_ERROR (0) EBADS (0x09) : Le numéro de socket n'est pas valide EPIPE (0x20) : la connexion est interrompue EWOULDBLOCK (0x23) : Le socket est saturé ECONNRESET (0x36) : La connexion a été réinitialisée par l'autre extrémité. ENOTCONN (0x39) : Le socket n'est pas connecté (socket d'entrée). INCORRECTLENGTH (0x0E) ; longueur > 240
Mngt.Length	En l'absence d'erreur, nombre d'octets transmis.

2.13 Fonction recv()

Syntaxe

```
#include <TCP/IP.h>
```

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

Description

La fonction recv() permet de recevoir des données du socket spécifié. Elle copie dans le tampon utilisateur les données disponibles sur le socket. La longueur maximale des données à recevoir est de 240 octets

La fonction recv() renvoie le nombre d'octets reçus ; cette valeur doit toujours être vérifiée, car c'est la seule façon de détecter le nombre véritable d'octets de données contenus dans le tampon utilisateur.

Il n'est pas possible de recevoir des données hors bande.

Paramètres reçus

Interface	Spécifie le numéro du module ETY5102
SocketNumber	Numéro du socket
pBuf	Pointe sur un tampon contenant les données
Mngt	Pointeur sur un tableau de quatre mots pour la gestion
Mngt.Length	Spécifie la taille du tampon, en octets. La longueur maximale est de 240 octets

Valeurs renvoyées

Mngt.SystemReport	signale une erreur si diffère de 0 (voir codes d'erreur au chap. 0)
Mngt.FunctionReport	<p>NO_ERROR (0)</p> <p>EBADS (0x09) : Le numéro de socket n'est pas valide</p> <p>EWOULDBLOCK (0x23) : Pas de données à lire</p> <p>ECONNRESET (0x36) : La connexion a été réinitialisée par l'autre extrémité.</p> <p>ENOTCONN (0x39) : Le socket n'est pas connecté (socket d'entrée)</p> <p>INCORRECTLENGTH (0x0E) ; longueur > 240</p> <p>ETIMEDOUT (0x3C) : Dépassement de délai du temporisateur de survie sur une connexion interrompue.</p>
Mngt.Length	En l'absence d'erreur, nombre d'octets stockés dans le tampon.

2.14 Fonction shutdown()

Syntaxe

```
#include <TCP/IP.h>
```

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

Description

Cette fonction permet de désactiver les transmissions et/ou les réceptions sur le socket. **shutdown()** peut être appelée pour désactiver la réception, la transmission ou les deux. Si how vaut 0, les réceptions ultérieures sur ce socket sont interdites.

La fenêtre TCP n'est pas modifiée, et les données reçues sont encore acceptées (mais sans accusé de réception) jusqu'à la fin de la fenêtre. Si how vaut 1, les transmissions ultérieures sont interdites. Un message FIN est alors envoyé. Si how vaut 2, les réceptions et les transmissions sont interdites comme décrit ci-dessus.

Notez que la fonction **shutdown()** ne ferme pas le socket ; les ressources affectées à ce socket ne sont libérées qu'à la suite de l'appel de **close()**. Toutefois, il est déconseillé de tenter de réutiliser un socket après l'exécution de la fonction **shutdown()**.

Paramètres reçus

Interface	Spécifie le numéro du module ETY5102
SocketNumber	Numéro du socket
how	spécifie le mécanisme d'arrêt. Voici les trois options disponibles : 0 les réceptions ne sont plus autorisées sur le socket 1 les transmissions ne sont plus autorisées sur le socket 2 les transmissions et les réceptions ne sont plus autorisées sur le socket
Mngt	Pointeur sur un tableau de quatre mots pour la gestion

Valeurs renvoyées

Mngt.SystemReport	signale une erreur si diffère de 0 (voir codes d'erreur au chap. 0)
Mngt.FunctionReport	NO_ERROR (0) EBADS (0x09) : Le numéro de socket n'est pas valide EINVAL (0x16) : un argument n'est pas valide. ENOTCONN (0x39) : Le socket n'est pas connecté

2.15 Fonction close()

Syntaxe

```
#include <TCP/IP.h>
```

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

Description

La fonction **close()** supprime le socket spécifié. Comme l'option SO_LINGER est définie avec un dépassement de délai de 0, **close()** n'est pas bloquée, même si les données en file d'attente n'ont pas encore été transmises ou n'ont pas fait l'objet d'un accusé de réception. Cette méthode est appelée fermeture "dure", car le circuit virtuel du socket est immédiatement réinitialisé, et toutes les données non transmises sont perdues. Tout appel à la fonction **recv()** à l'autre extrémité du circuit renverra le message d'erreur CONNRESET.

Paramètres reçus

Interface	Spécifie le numéro du module ETY5102
SocketNumber	Numéro du socket
Mngt	Pointeur sur un tableau de quatre mots pour la gestion

Output

Mngt.SystemReport	signale une erreur si diffère de 0 (voir codes d'erreur au chap. 0)
Mngt.FunctionReport	NO_ERROR (0) EINVAL (0x16) : Le numéro de socket n'est pas valide

Important : Si SocketNumber est égal à 0, la fonction supprime **tous les sockets** Du profil OPEN.

Fonctions élémentaires de niveau 1 pour programmation avancée

Taille en octets des fonctions élémentaires générées :

Nom de la EF	Taille en octets
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

Le module ETY5102 possède quatre états de fonctionnement : arrêt, phase d'autotest, non configuré et configuré. Après la mise sous tension, le module effectue ses autotests. Le module ne fonctionne pas sans configuration par défaut. La configuration doit être transmise par l'application PL7 du PLC local. Lorsque l'état de fonctionnement n'est pas celui qui a été configuré, le module n'est pas en mesure de recevoir les opérations des sockets ; un message de refus est alors renvoyé à la bibliothèque de sockets.

La configuration est transmise par l'application PL7 dans l'ordre suivant :

- lors du chargement d'une nouvelle application
- après la mise sous tension ou la déconnexion du module du rack
- après un redémarrage à chaud
- lors de la réinitialisation de Premium.

Après avoir reçu la configuration, le module réinitialise la communication en cours avant de s'auto-configurer (les échanges en cours sont interrompus, les connexions TCP ouvertes sont fermées, et tous les sockets sont supprimés).

Le programmeur doit ensuite tester les bits système S0 et S1 dans son application, et recréer les connexions si un redémarrage à chaud ou à froid est détecté.

Les transitions marche/arrêt et arrêt/marche sont sans effet sur l'état des sockets.

Le programmeur est responsable du test d'état des connexions (lorsqu'elles sont fermées). Un client qui a mis fin à une connexion sans respecter la séquence TCP correcte (ex. : mise hors tension) est perçu comme étant toujours connecté tant qu'il n'y a pas eu d'opération send() ou tout autre événement (ex. : séquence KEEP_ALIVE). Un appel à recv() est sans effet sur le réseau, et ne peut pas provoquer de détection de déconnexion (voir ci-dessous "Notes relatives aux applications").

Dans la fenêtre de débogage de PL7, le nombre de connexions affiché inclut les connexions de profils privés et ouverts.

Les adresses IP de clients de profils ouverts ne sont pas affichées dans cette fenêtre.

Test de communication IP

Il est possible d'utiliser la fenêtre de test de communication pour tester les communications IP avec des équipements clients si l'adresse IP du client est déclarée parmi les équipements distants (utilisés par le profil privé). La liste des adresses IP configurées permet de sélectionner la station avec laquelle l'utilisateur désire communiquer, en utilisant l'utilitaire "ping", qui permet de connaître le délai de réponse au message (ou le dépassement de ce délai).

6.1 Nombre de connexions

Le nombre maximal de connexions autorisées avec OPEN TCP/IP est de 16. Ce nombre peut être inférieur à 16 si le profil TCP/IP privé utilise plus de 16 connexions, car le nombre total de connexions est de 32 au maximum pour l'ETY5102, et le profil TCP/IP privé peut utiliser jusqu'à 32 connexions, selon la configuration.

6.2 Échange de données

Les connexions TCP/IP transmettent des flux de données par octets. Il est possible à tout moment d'ajouter des informations à ces flux de données. En raison des limitations du mécanisme de transport X-Bus, la taille maximale de chaque bloc de données est limitée à 240 octets par cycle PLC si l'utilisateur désire préserver l'ordre séquentiel de transmission des informations. Un message de 8 kOctets nécessite 35 cycles PLC ($8 \times 1024 / 240$) en transmission ou en réception. Cela correspond à un délai de 1,75 seconde pour un cycle de 50ms, ceci pour un seul socket.

Pour un protocole orienté messages, il est nécessaire qu'une interface de bas niveau prenne en charge le processus de fragmentation. À ce niveau, les performances dépendent du nombre d'exécutions des fonctions send() ou receive() au cours d'un même cycle PLC.

Ces performances peuvent être inférieures, en fonction du taux d'utilisation du module ETY5102 pour d'autres tâches de communications sur des profils privés, des profils ETHWAY ou CWE (Common Words Exchanges).

7.1 Fonctions élémentaires de niveau 1 pour programmation avancée

7.1.1 EF : FCT_SOCKET_DFB

7.1.1.1 Description

La fonction élémentaire (EF) FCT_SOCKET est une encapsulation de la fonction socket() qui permet d'utiliser celle-ci avec le langage PL7. Elle crée un nouveau socket et renvoie son numéro.

7.1.1.2 Source de la fonction dans le SDKC

```
//PL7SDK_BEGIN_FUNCTION_NAME
// ne pas modifier le prototype
// La fonction EF principale doit être la première de ce
fichier
#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, //Numéro de module
short IGST, //Index du premier mot de gestion dans le
tableau GEST
adrTABLE SOCK, // Numéro de socket retourné
adrTABLE GEST //Paramètres de gestion
)
//PL7SDK_END_FUNCTION_NAME
{
    #include <TCP/IP.h>

    Mngt far *pParametreGestion;
    unsigned short far *pSocket;

    /* calcul de l'adresse physique de l'adresse du tableau de
    gestion */
    pParametreGestion = (Mngt *)GET_ADDR (GEST.selecteur,
    GEST.offset);
    pSocket = (unsigned short *)GET_ADDR (SOCK.selecteur,
    SOCK.offset);

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

7.1.1.3 Interface

Paramètres reçus :

Nom	Type	Commentaire
INTE	WORD	Numéro de module
IGST	WORD	Index du premier mot dans le tableau GEST

Paramètres renvoyés :

Nom	Type	Commentaire
SOCK	WORD	Numéro de socket retourné

Paramètres d'E/S :

Nom	Type	Commentaire
GEST	AR_W	Paramètre de Gestion

7.1.1.4 Programmation PL7

En langage ST, cette EF sera appelée ainsi :

(* ouvrir socket sur module numéro 4 *)

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

7.1.2 EF : FCT_BIND_DFB

7.1.2.1 Description

La fonction élémentaire FCT_BIND est une encapsulation de la fonction bind() qui permet d'utiliser celle-ci avec le langage PL7. Cette fonction élémentaire permet d'attribuer (ou relier) à un socket une adresse Internet et un numéro de port.

7.1.2.2 Source de la fonction dans le SDKC

```
//PL7SDK_BEGIN_FUNCTION_NAME
// ne pas modifier le prototype
// La fonction EF principale doit être la première de ce
fichier
#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,//Numéro de module
short SOCK,//Numéro de socket
short PORT,//Numéro de port
short IGST,//Index du premier mot de gestion dans le
tableau GEST
adrTABLE GEST//Paramètres de gestion
)
//PL7SDK_END_FUNCTION_NAME
{
    #include <TCP/IP.h>
    Mngt far *pParametreGestion;

/* calcul de l'adresse physique de l'adresse du tableau de
gestion */
    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

Paramètres reçus :

Nom	Type	Commentaire
INTE	WORD	Numéro de module
SOCK	WORD	Numéro de socket
PORT	WORD	Numéro du port
IGST	WORD	Index du premier mot dans le tableau GEST

Paramètres renvoyés :

Nom	Type	Commentaire

Paramètres d'E/S :

Nom	Type	Commentaire
GEST	AR_W	Paramètre de Gestion

7.1.2.4 Programmation PL7

En langage ST, cette EF sera appelée ainsi :

(*Relier le socket numéro 5 du module numéro 4 au numéro de port 505*)

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

7.1.3 EF : FCT_CONNECT_DFB

7.1.3.1 Description

La fonction élémentaire FCT_CONNECT_DFB est une encapsulation de la fonction connect() qui permet d'utiliser celle-ci avec le langage PL7. Cette EF permet d'établir une connexion à un port et une adresse Internet connus.

7.1.3.2 Interface

Paramètres reçus :

Nom	Type	Commentaire
INTE	WORD	Numéro de module
SOCK	WORD	Numéro de socket
ISRV	WORD	Index du premier mot dans le tableau SERV
IGST	WORD	Index du premier mot dans le tableau GEST

Paramètres renvoyés :

Nom	Type	Commentaire

Paramètres d'E/S :

Nom	Type	Commentaire
SERV	AR_W	Port + Adresse IP du serveur
GEST	AR_W	Paramètre de Gestion

7.1.3.3 Programmation PL7

En langage ST, cette EF sera appelée ainsi :

(*Connecter le socket numéro 5 du module numéro 4 au serveur *)

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

7.1.4 EF : FCT_LISTEN_DFB

7.1.4.1 Description

La fonction élémentaire EF FCT_LISTEN est une encapsulation de la fonction listen() qui permet d'utiliser celle-ci avec le langage PL7. Cette fonction configure le socket spécifié pour recevoir des connexions.

7.1.4.2 Source de la fonction dans le SDKC

```
//PL7SDK_BEGIN_FUNCTION_NAME
// ne pas modifier le prototype
// La fonction EF principale doit être la première de ce
fichier
#include <Cstsys.h>
#include <Fctsys.h>
#include <C:\ASAWTEMP\SDKC\CLASSE05.H >
#include <C:\ASAWTEMP\SDKC\MAIN0501.H>
void far pascal FCT_LISTEN(
short INTE,//Numéro de module
short SOCK,//Numéro de socket
short IGST,//Index du premier mot de gestion dans le
tableau GEST
adrTABLE GEST//Paramètres de gestion
)
//PL7SDK_END_FUNCTION_NAME
{
    #include <TCP/IP.h>
    Mngt far *pParametreGestion;

    /* calcul de l'adresse physique de l'adresse du tableau de
gestion */
    pParametreGestion = (Mngt *)GET_ADDR (GEST.selecteur,
GEST.offset);

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

7.1.4.3 Interface

Paramètres reçus :

Nom	Type	Commentaire
INTE	WORD	Numéro de module
SOCK	WORD	Numéro de socket
IGST	WORD	Index du premier mot dans le tableau GEST

Paramètres renvoyés :

Nom	Type	Commentaire

Paramètres d'E/S :

Nom	Type	Commentaire
GEST	AR_W	Paramètre de Gestion

7.1.4.4 Programmation PL7

En langage ST, cette EF sera appelée ainsi :

(* écouter socket numéro %MW10 sur module numéro 4 *)

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

7.1.5 EF : FCT_ACCEPT_DFB

7.1.5.1 Description

La fonction élémentaire EF FCT_ACCEPT est une encapsulation de la fonction accept() qui permet d'utiliser celle-ci avec le langage PL7. Cette fonction permet d'accepter une demande de connexion reçue par le socket spécifié depuis un socket extérieur.

7.1.5.2 Source de la fonction dans le SDKC

```
//PL7SDK_BEGIN_FUNCTION_NAME
// ne pas modifier le prototype
// La fonction EF principale doit être la première de ce
fichier
#include <Cstsyst.h>
#include <Fctsyst.h>
#include <C:\ASAWTEMP\SDKC\CLASSE06.H >
#include <C:\ASAWTEMP\SDKC\MAIN0601.H>
void far pascal FCT_ACCEPT(
short INTE, //Numéro de module
short SOCK, //Numéro de socket
short ICLIE, //Index du premier mot dans le tableau
CLIE
short IGST, //Index du premier mot de gestion dans le tableau
GEST
adrTABLE CLIE, //Socket service + port + IP du client
adrTABLE GEST //Paramètre de gestion
)
//PL7SDK_END_FUNCTION_NAME
{
    #include <TCP/IP.h>

    Mngt far *pParametreGestion;
    unsigned short far *pParametreClient;

    /* calcul de l'adresse physique de l'adresse du tableau de
gestion */
    pParametreClient = (unsigned short *)GET_ADDR
(CLIE.selecteur, CLIE.offset);
    /* calcul de l'adresse physique du tableau de gestion */
    pParametreGestion = (Mngt *)GET_ADDR (GEST.selecteur,
GEST.offset);

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

7.1.5.3 Interface

Paramètres reçus :

Nom	Type	Commentaire
INTE	WORD	Numéro de module
SOCK	WORD	Numéro de socket
ICLIE	WORD	Index du premier mot dans le tableau CLIE
IGST	WORD	Index du premier mot dans le tableau GEST

Paramètres renvoyés :

Nom	Type	Commentaire
CLIE	AR_W	Socket de service + port + IP client

Paramètres d'E/S :

Nom	Type	Commentaire
GEST	AR_W	Paramètre de Gestion

7.1.5.4 Programmation PL7

En langage ST, cette EF sera appelée ainsi :

(* accepter connexion sur le socket %MW10 sur module numéro 4 *)

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

7.1.6 EF : FCT_SEND_DFB

7.1.6.1 Description

La fonction élémentaire EF FCT_SEND est une encapsulation de la fonction send() qui permet d'utiliser celle-ci avec le langage PL7. Cette fonction permet d'envoyer des données à un socket extérieur. La longueur maximale des données à transmettre est de 240 octets

7.1.6.2 Source de la fonction dans le SDKC

```
//PL7SDK_BEGIN_FUNCTION_NAME
// ne pas modifier le prototype
// La fonction EF principale doit être la première de ce
fichier
#include <Cstsyst.h>
#include <Fctsyst.h>
#include <C:\ASAWTEMP\SDKC\CLASSE01.H >
#include <C:\ASAWTEMP\SDKC\MAIN0101.H>
void far pascal FCT_SEND(
short INTE, //Numéro de module
short SOCK, //Numéro de socket
short IBUF, //Index du premier caractère dans PBUF
short IGST, //Index du premier mot de gestion dans le tableau
GEST
adrTABLE PBUF, //Données à envoyer
adrTABLE GEST //Paramètres de gestion
)
//PL7SDK_END_FUNCTION_NAME
{
#include <TCP/IP.h>

    char far *pBuffer;
    Mngt far *pParametreGestion;

    /* calcul de l'adresse physique de l'adresse du buffer
d'emission */
    pBuffer= (char *)GET_ADDR (PBUF.selecteur, PBUF.offset);
    /* calcul de l'adresse physique de l'adresse du tableau de
gestion */
    pParametreGestion = (Mngt *)GET_ADDR (GEST.selecteur,
GEST.offset);

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

7.1.6.3 Interface

Paramètres reçus :

Nom	Type	Commentaire
INTE	WORD	Numéro de module
SOCK	WORD	Numéro de socket
IBUF	WORD	Index du premier mot dans le tableau PBUF
IGST	WORD	Index du premier mot dans le tableau GEST
PBUF	AR_W	Données à envoyer

Paramètres renvoyés :

Nom	Type	Commentaire

Paramètres d'E/S :

Nom	Type	Commentaire
GEST	AR_W	Paramètre de Gestion

7.1.6.4 Programmation PL7

En langage ST, cette EF sera appelée ainsi :

(* transmettre données sur socket numéro %MW10 du module numéro 4 *)

FCT_SEND_DFB(4,%MW10,%MW100:120,%MW0:4);

7.1.7 EF : FCT_RECEIVE_DFB

7.1.7.1 Description

La fonction élémentaire FCT_RECEIVE est une encapsulation de la fonction `recv()` qui permet d'utiliser celle-ci avec le langage PL7. Elle copie dans le tampon utilisateur les données disponibles sur le socket. La longueur maximale des données à recevoir est de 240 octets

7.1.7.2 Source de la fonction dans le SDKC

```
//PL7SDK_BEGIN_FUNCTION_NAME
// ne pas modifier le prototype
// La fonction EF principale doit être la première de ce
fichier
#include <Cstsyst.h>
#include <Fctsyst.h>
#include <C:\ASAWTEMP\SDKC\CLASSE02.H >
#include <C:\ASAWTEMP\SDKC\MAIN0201.H>
void far pascal FCT_RECEIVE(
short INTE, //Numéro de module
short SOCK, //Numéro de socket
short IBUF, //Index du premier caractère dans PBUF
short IGST, //Index du premier mot de gestion dans le tableau
GEST
adrTABLE PBUF, //Données recues
adrTABLE GEST //Pointeur de gestion
)
//PL7SDK_END_FUNCTION_NAME
{
    #include <TCP/IP.h>

    char far *pBuffer;
    Mngt far *pParametreGestion;

    /* calcul de l'adresse physique de l'adresse du buffer
d'emission */
    pBuffer = (char *)GET_ADDR (PBUF.selecteur, PBUF.offset);

    /* calcul de l'adresse physique de l'adresse du tableau de
gestion */
    pParametreGestion = (Mngt *)GET_ADDR (GEST.selecteur,
GEST.offset);

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

7.1.7.3 Interface

Paramètres reçus :

Nom	Type	Commentaire
INTE	WORD	Numéro de module
IBUF	WORD	Index du premier mot dans le tableau PBUF
SOCK	WORD	Numéro de socket
IGST	WORD	Index du premier mot dans le tableau GEST

Paramètres renvoyés :

Nom	Type	Commentaire
PBUF	AR_W	Données reçues

Paramètres d'E/S :

Nom	Type	Commentaire
GEST	AR_W	Paramètre de Gestion

7.1.7.4 Programmation PL7

En langage ST, cette EF sera appelée ainsi :

(* recevoir données sur socket numéro %MW13 du module numéro 4 *)

FCT_RECEIVE_DFB(4,%MW30,%MW100:120,%MW0:4);

7.1.8 EF : FCT_STCKOPT_DFB

7.1.8.1 Description

La fonction élémentaire FCT_SETSOCKOPT est une encapsulation de la fonction setsockopt() qui permet d'utiliser celle-ci avec le langage PL7. Cette fonction configure les options associées au socket spécifié. Le nombre d'options est volontairement réduit en raison du mode de fonctionnement de l'ETY5102. Certaines options sont définies lors de la création du socket. Voir la description de la fonction setsockopt().

7.1.8.2 Source de la fonction dans le SDKC

```
//PL7SDK_BEGIN_FUNCTION_NAME
// ne pas modifier le prototype
// La fonction EF principale doit être la première de ce
fichier
#include <Cstsyst.h>
#include <Fctsyst.h>
#include <C:\ASAWTEMP\SDKC\CLASSE07.H >
#include <C:\ASAWTEMP\SDKC\MAIN0701.H>
void far pascal FCT_SETSOCKOPT(
short INTE,//Numéro de module
short SOCK,//Numéro de socket
short OPT,//Type d'option
short IGST,//Index du premier mot de gestion dans le
tableau GEST
adrTABLE GEST//Paramètres de gestion
)
//PL7SDK_END_FUNCTION_NAME
{
    #include <TCP/IP.h>

    Mngt far *pParametreGestion;

    /* calcul de l'adresse physique de l'adresse du tableau de
gestion */
    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

Paramètres reçus :

Nom	Type	Commentaire
INTE	WORD	Numéro de module
SOCK	WORD	Numéro de socket
OPT	WORD	Type d'option
IGST	WORD	Index du premier mot dans le tableau GEST

Paramètres renvoyés :

Nom	Type	Commentaire

Paramètres d'E/S :

Nom	Type	Commentaire
GEST	AR_W	Paramètre de Gestion

7.1.8.4 Programmation PL7

En langage ST, cette EF sera appelée ainsi :

(* activer l'option DONT_ROUTE sur socket numéro %MW10 du module numéro 4 *)

%MW20:= DONT_ROUTE;

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

7.1.9 EF : FCT_SELECT_DFB

7.1.9.1 Description

La fonction élémentaire EF FCT_SELECT est une encapsulation de la fonction select() qui permet d'utiliser celle-ci avec le langage PL7. Cette fonction permet de multiplexer des demandes d'E/S sur plusieurs sockets. Un masque de bits de deux mots est renvoyé par la fonction ; il indique les sockets qui ont des événements en attente de traitement.

7.1.9.2 Source de la fonction dans le SDKC

```
//PL7SDK_BEGIN_FUNCTION_NAME
// ne pas modifier le prototype
// La fonction EF principale doit être la première de ce
fichier
#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, //Numéro de module
short IMASK, //Index du premier caractère dans MASK
short IGST, //Index du premier mot de gestion dans le tableau
GEST
adrTABLE MASK, //Etat d'activité de chaque socket
adrTABLE GEST //Paramètres de gestion
)
//PL7SDK_END_FUNCTION_NAME
{
    #include <TCP/IP.h>

    Mngt far *pParametreGestion;
    unsigned short far *pMask;

    /* calcul de l'adresse physique de l'adresse du tableau de
gestion */
    pParametreGestion = (Mngt *)GET_ADDR (GEST.selecteur,
GEST.offset);
    /* calcul de l'adresse physique de l'adresse du tableau de
gestion */
    pMask = (unsigned short *)GET_ADDR (MASK.selecteur,
MASK.offset);

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

7.1.9.3 Interface

Paramètres reçus :

Nom	Type	Commentaire
INTE	WORD	Numéro de module
IMASK	WORD	Index du premier mot dans le tableau MASK
IGST	WORD	Index du premier mot dans le tableau GEST

Paramètres renvoyés :

Nom	Type	Commentaire
MASK	AR_W	Etat d'activité de chaque socket

Paramètres d'E/S :

Nom	Type	Commentaire
GEST	AR_W	Paramètre de Gestion

7.1.9.4 Programmation PL7

En langage ST, cette EF sera appelée ainsi :

(* fonction sélection sur module numéro 4 *)

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

7.1.10 EF : FCT_SHUTDOWN_DFB

7.1.10.1 Description

La fonction élémentaire FCT_SHUTDOWN est une encapsulation de la fonction shutdown() qui permet d'utiliser celle-ci avec le langage PL7. Cette fonction permet de désactiver la transmission et/ou la réception sur le socket.

7.1.10.2 Source de la fonction dans le SDKC

```
//PL7SDK_BEGIN_FUNCTION_NAME
// ne pas modifier le prototype
// La fonction EF principale doit être la première de ce
fichier
#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,//Numéro de module
short SOCK,//Numéro de socket
short HOW,//Option de shutdown
short IGST,//Index du premier mot de gestion dans le
tableau GEST
adrTABLE GEST//Paramètres de gestion
)
//PL7SDK_END_FUNCTION_NAME
{
    #include <TCP/IP.h>

    Mngt far *pParametreGestion;

    /* calcul de l'adresse physique de l'adresse du tableau de
gestion */
    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

Paramètres reçus :

Nom	Type	Commentaire
INTE	WORD	Numéro de module
SOCK	WORD	Numéro de socket
HOW	WORD	Option de shutdown
IGST	WORD	Index du premier mot dans le tableau GEST

Paramètres renvoyés :

Nom	Type	Commentaire

Paramètres d'E/S :

Nom	Type	Commentaire
GEST	AR_W	Paramètre de Gestion

7.1.10.4 Programmation PL7

En langage ST, cette EF sera appelée ainsi :

(* arrêter socket numéro %MW10 du module numéro 4 *)

%MW20 :=2; (*transmissions et réceptions plus autorisées sur le socket*)

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

7.1.11 EF : FCT_CLOSE_DFB

7.1.11.1 Description

La fonction élémentaire EF FCT_CLOSE est une encapsulation de la fonction `close()` qui permet d'utiliser celle-ci avec le langage PL7. Cette fonction supprime le socket spécifié. Si `SocketNumber` est égal à 0, la fonction supprime **tous les sockets** du profil OPEN.

7.1.11.2 Source de la fonction dans le SDKC

```
//PL7SDK_BEGIN_FUNCTION_NAME
// ne pas modifier le prototype
// La fonction EF principale doit être la première de ce
fichier
#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,//Numéro de module
short SOCK,//Numéro de socket
short IGST,//Index du premier mot de gestion dans le
tableau GEST
adrTABLE GEST//Paramètres de gestion
)
//PL7SDK_END_FUNCTION_NAME
{
    #include <TCP/IP.h>

    Mngt far *pParametreGestion;

    /* calcul de l'adresse physique de l'adresse du tableau de
gestion */
    pParametreGestion = (Mngt *)GET_ADDR (GEST.selecteur,
GEST.offset);

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

7.1.11.3 Interface

Paramètres reçus :

Nom	Type	Commentaire
INTE	WORD	Numéro de module
SOCK	WORD	Numéro de socket
IGST	WORD	Index du premier mot dans le tableau GEST

Paramètres renvoyés :

Nom	Type	Commentaire

Paramètres d'E/S :

Nom	Type	Commentaire
GEST	AR_W	Paramètre de Gestion

7.1.11.4 Programmation PL7

En langage ST, cette EF sera appelée ainsi :

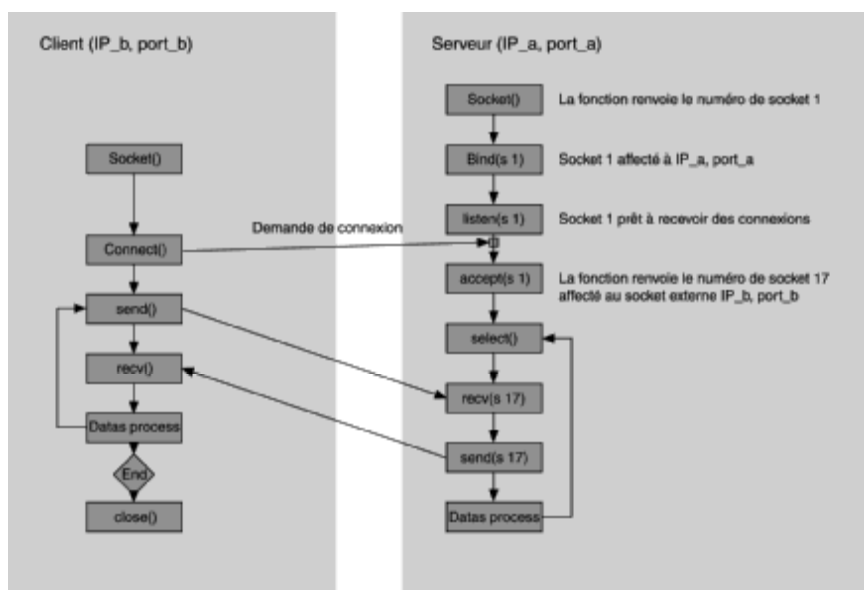
(* fermer socket numéro %MW10 du module numéro 4 *)

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

7.2 Modèle client/serveur

Le client et le serveur nécessitent un ensemble de conventions courantes pour que le service puisse être effectué et accepté. Cet ensemble de conventions se compose d'un protocole qui doit être implémenté de part et d'autre de la connexion. Le programmeur est responsable du test d'état des connexions.

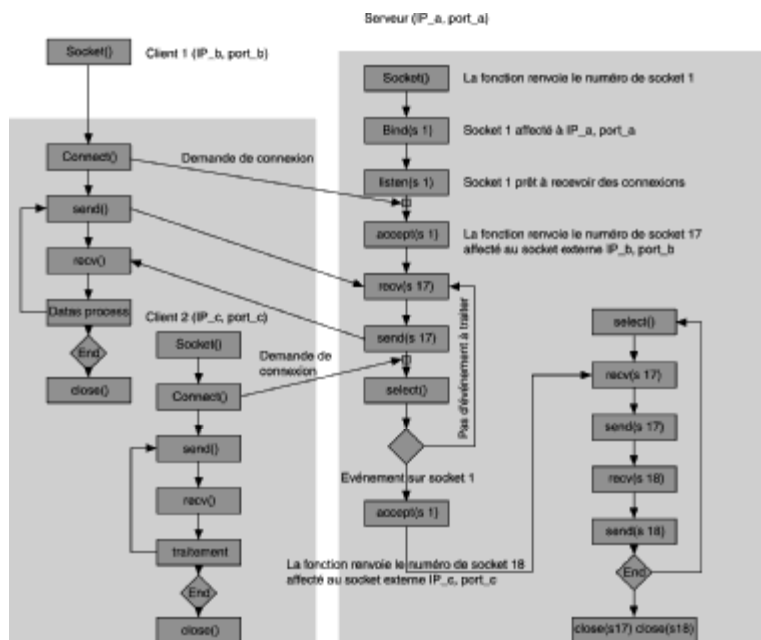
L'application serveur est en écoute (attente de demandes de service). Suivant ce modèle, les applications clientes demandent des services à une application serveur. L'établissement des communications entre le client et le serveur est donc asymétrique.



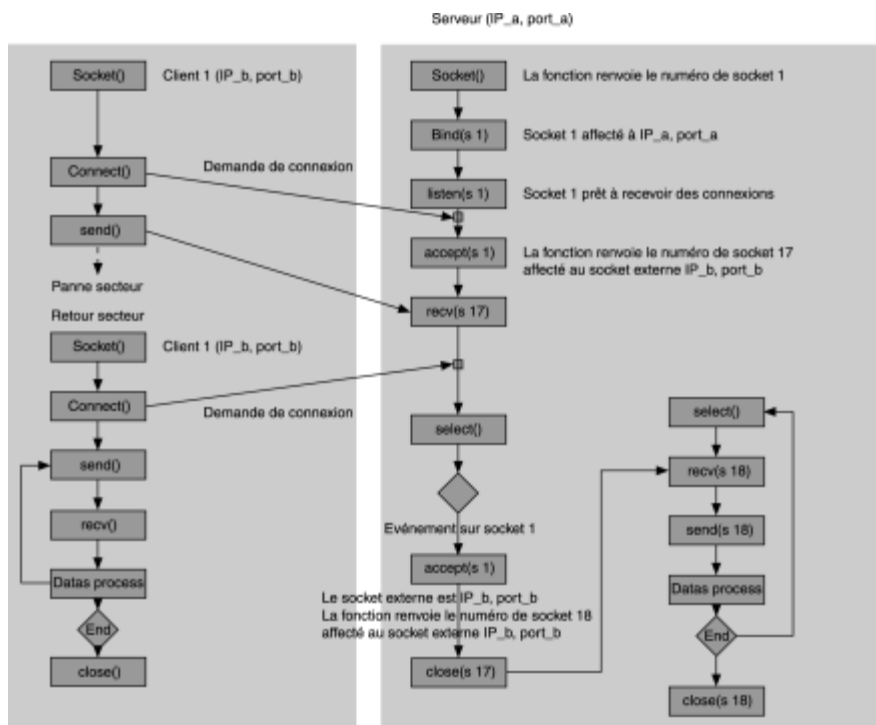
7.3 Client/serveur : modes de fonctionnement

Le programmeur est responsable du test d'état des connexions. Un client qui a mis fin à une connexion sans respecter la séquence TCP correcte (ex. : mise hors tension) est perçu comme étant toujours connecté tant qu'il n'y a pas eu d'opération `send()` ou tout autre événement (ex. : séquence `KEEP_ALIVE`). Un appel à `recv()` est sans effet sur le réseau, et ne peut pas provoquer de détection de déconnexion.

Exemple 1 : L'application serveur traite deux connexions demandées par deux clients

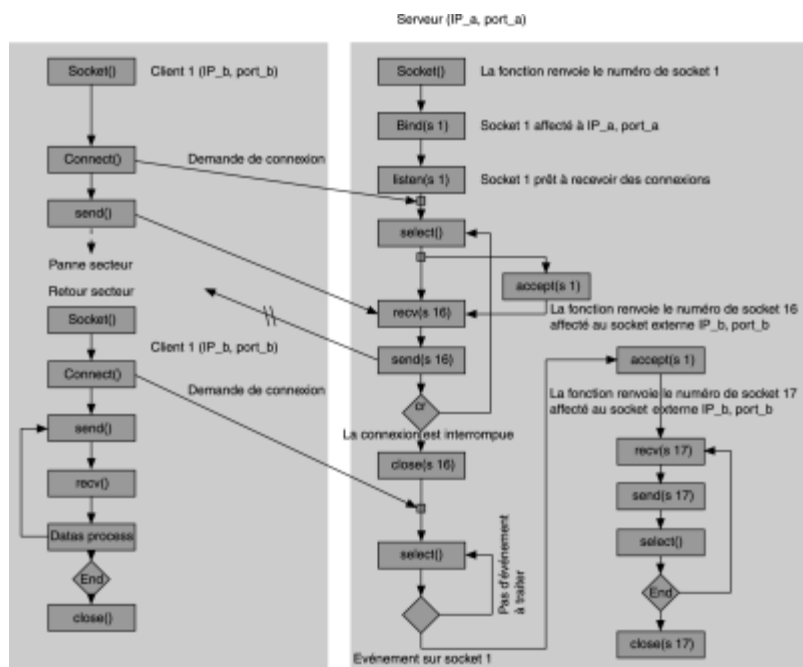


Exemple 2 : L'application serveur traite deux connexions demandées par le même client. Le client a mis fin à une connexion sans respecter la séquence TCP correcte (ex. : panne secteur). Le socket est perçu par le serveur comme étant toujours connecté tant qu'il n'y a pas eu de nouvelle opération connect() demandée par le client.



Remarque : Si le protocole DHCP (*Dynamic Host Configuration Protocol*) n'est pas utilisé, le serveur peut comparer les adresses IP des clients.

Exemple 3 : L'application serveur traite deux connexions demandées par le même client. Le client a mis fin à une connexion sans respecter la séquence TCP correcte (ex. : panne secteur). Le socket est perçu par le serveur comme étant toujours connecté tant qu'il n'y a pas eu d'opération `send()` demandée par le client.



GLOSSAIRE**Serveur :**

Dans le contexte de TCP/IP, le serveur est le nœud qui attend des connexions.

Client :

Dans le contexte de TCP/IP, le client est le nœud qui demande une connexion.

ABRÉVIATIONS**EF :**

Fonction élémentaire (Elementary Function).

Chapitre	Page
1 Présentation générale	1/1
1.1 Généralités	1/1
2 Installation	2/1
2.1 Procédure d'installation	2/1
2.2 Configuration du coupleur TSX ETY 110 WS/5102	2/2
2.3 Programmation des DFBs dans une application	2/3
2.3-1 Import des DFBs	2/3
2.3-2 Création d'une instance de DFB	2/3
3 Principes de mise en œuvre	3/1
3.1 Principes d'une connexion TCP	3/1
3.2 Principe de base des DFBs	3/2
3.2-1 Etablissement de connexions avec sécurité d'accès	3/2
3.2-2 Transfert de messages	3/5
4 Interfaces avec l'application PL7	4/1
4.1 Les points communs des DFBs	4/1
4.1-1 L'appel entretenu	4/1
4.1-2 La table de gestion	4/2
4.1-3 Les bits RST et ACTIVITY	4/4
4.1-4 Le bit ERROR - mot STATUS	4/5
5 Gestion des connexions DFB TCP_CNx	5/1
5.1 Présentation	5/1

Chapitre	Page
5.1-1 Caractéristiques	5/1
5.1-2 Fonctionnement	5/4
5.1-3 Exemple de programmation - Connexion sur 1 seul port TCP	5/5
6 Emission de données DFB TCP_SEND	6/1
6.1 Présentation	6/1
6.1-1 Caractéristiques	6/1
6.1-2 Fonctionnement	6/4
6.1-3 Exemple de programmation	6/5
7 Réception de données DFB TCP_RECEIVE	7/1
7.1 Présentation	7/1
7.1-1 Caractéristiques	7/1
7.1-2 Fonctionnement	7/4
7.1-3 Exemple de programmation	7/5
8 Performances	8/1
8.1 Points intervenants dans les performances d'une communication TCPIP-OPEN	8/1
8.2 Schéma d'un traitement d'EF dans un cycle automate	8/2
8.3 Mesures de performances	8/2
9 Annexes	9/1
9.1 Différences avec les OFB PL7-3	9/1
9.2 Rappel sur la communication TCPIP	9/2

1.1 Généralités

La bibliothèque des DFBs de communication TCP utilisée avec le coupleur TSX ETY 110 WS/5102 version ≥ 2.8 pour automates Premium permet le transfert de bloc de données entre une application automate et une application distante sur une connexion TCPIP établie à l'initiative de l'application distante.

L'application distante est exécutée sur un équipement supportant le profil de communication TCPIP.

La bibliothèque des DFBs de communication se compose :

- d'un DFB TCP_CNX de gestion des connexions,
- d'un DFB TCP_SEND pour l'émission de bloc de données d'une taille maximale de 8 K octets,
- d'un DFB TCP_RECEIVE pour la réception de bloc de données d'une taille maximale de 8 K octets.

Note:

Les DFBs de communication TCP pour automates Premium utilisent les services de l'offre "OPEN TCP for TSX Premium" qui est elle même constituée d'une bibliothèque d'EF (Elementary Function) TCP: EF permettant à l'application PL7 de gérer des connexions TCP et d'émettre et recevoir des flux d'octets sur ces connexions.

L'installation du produit "DFBs de messagerie libre TCPIP" complète un atelier de programmation PL7 Pro de version \geq à V3.3 en installant :

- 3 DFBs TCP_CNx, TCP_SEND, TCP_RECEIVE,
- et une famille d'EF TCPIP_DFB.

2.1 Procédure d'installation

Le CD-ROM TLXCDUNTCPB33F fournit l'accès à la documentation et à la procédure d'installation.

Insérer le CD-ROM, valider l'écran de bienvenue et laissez vous guider par les menus.

2.2 Configuration du coupleur TSX ETY 110 WS/5102

La configuration du coupleur TSX ETY 110 WS/5102 est réalisée par PL7 en fonction des données saisies par l'utilisateur dans l'écran de configuration du coupleur :

- Adresse IP locale de l'automate sur le réseau.
- Le masque de sous-adressage (SubNet Mask).
- L'adresse IP de la gateway par défaut.
- L'utilisation des trames Ethernet 802.3 avec SNAP (SubNetwork Access Protocol).

Le coupleur TSX ETY 110 WS/5102 doit être installé dans le rack 0.

2.3 Programmation des DFBs dans une application

2.3-1 Import des DFBs

Sous PL7 Pro, dans la "Vue Structurelle", et sous la rubrique "Types DFB", sélectionnez "Importer binaire".

Pour chacun des 3 DFBs TCP_CNX, TCP_RECEIVE, TCP_SEND les importer dans l'application.

2.3-2 Création d'une instance de DFB

Sous PL7 Pro sélectionner "Outils", "Bibliothèque" puis l'onglet "DFB".

Choisir ensuite le type de DFB que vous désirez instancier et valider par le bouton "Créer".

Saisir ensuite le nom d'instance de votre DFB et valider par le bouton "Créer".

Votre DFB est alors instancié. C'est ce nom d'instance qui est utilisé dans le programme automate PL7.

ATTENTION

L'import des DFBs n'est possible que si la procédure d'installation du CD-ROM TLXCDUNTCPB33F a été déroulée entièrement :

- installation des DFBs TCP_CNX, TCP_RECEIVE, TCP_SEND,
- installation de famille d'EF TCPIP_DFB dans la bibliothèque.

3.1 Principes d'une connexion TCP

L'établissement de la connexion entre deux applications est asymétrique, elle est basée sur un modèle client/serveur.

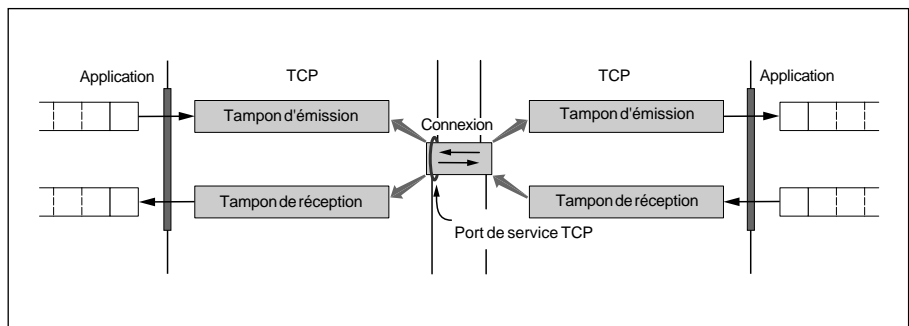
Pour qu'une connexion puisse être établie entre deux applications, cela nécessite d'avoir une application serveur en attente de connexion entrante sur un port de service TCP.

Le protocole TCP utilise des numéros de ports pour identifier les destinataires sur une machine donnée. Lorsque le numéro de port est identifié par le serveur, le client peut alors envoyer une demande de connexion vers l'application serveur.

L'application serveur accepte la connexion entrante. Une fois la connexion ouverte, un tampon d'émission et un tampon de réception sont attribués à ce port de service TCP.

Sur une connexion établie, le transfert de données peut alors être effectué.

Le transfert de données est fiable, (données stockées en zone tampon), non structuré (flots d'octets et non de messages) et bidirectionnel simultané (full-duplex).



3.2 Principe de base des DFBs

3.2-1 Etablissement de connexions avec sécurité d'accès

L'automate serveur se met en attente de connexions entrantes pour chaque port local TCP de service.

Cette mise en attente de connexion est gérée par le DFB TCP_CNX. Toutes les machines distantes déclarées dans la configuration du DFB TCP_CNX peuvent donc se connecter à l'automate. Juste après acceptation d'une connexion entrante sur un port local TCP de service, le DFB TCP_CNX vérifie que l'adresse IP de la machine distante appartient à la liste des machines distantes autorisées à se connecter.

Si l'adresse IP distante est référencée dans cette liste, la connexion sur le port local TCP de service est acceptée. Cette connexion est ensuite mémorisée dans la liste des connexions ouvertes. Le port local TCP de service associé est occupé, il n'est pas accessible pour toute autre demande de connexion.

Si l'adresse IP distante n'est pas référencée dans la liste des adresses autorisées, le DFB TCP_CNX ferme la connexion et repasse en attente de connexion sur ce port local TCP de service.

Remarque

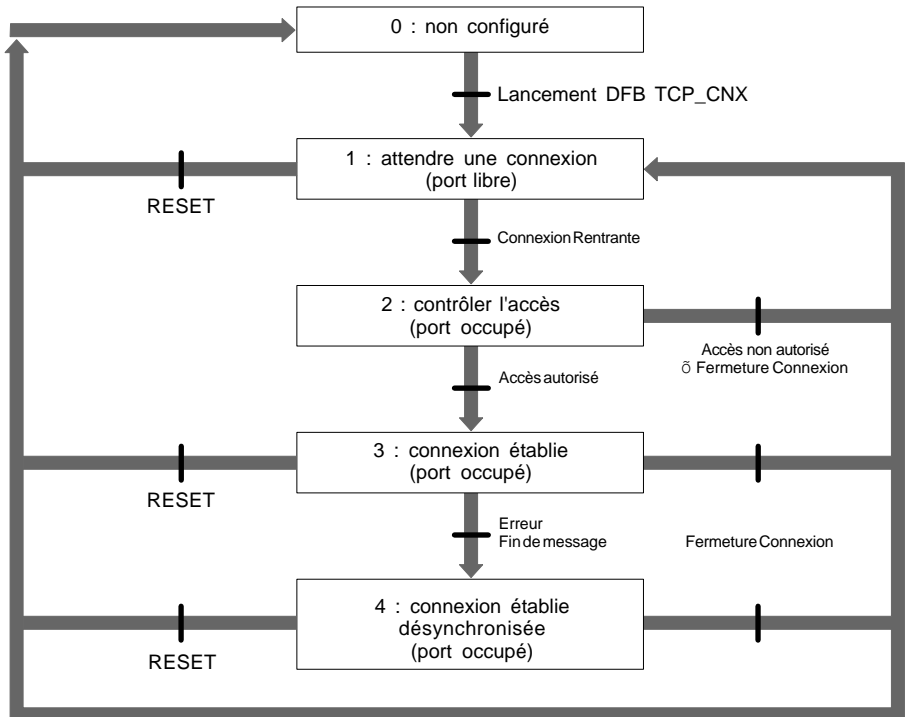
Une connexion sera immédiatement refermée par le DFB TCP_CNX pour une des raisons suivantes :

- Port occupé par une autre machine (une connexion par port).
- Adresse IP non autorisée.

Chaque connexion TCP ouverte avec le coupleur est donc caractérisée de façon unique par le port local TCP de service (1 port ÷ 1 connexion).

L'adresse IP de la machine distante est uniquement un attribut associé à cette connexion.

Graphe d'état d'une connexion pour un port

**Note :**

Ce graphe d'état, géré par le DFB TCP_CNX est pour un port TCP de service. Nous retrouvons ce même graphe d'état pour chacun des autres ports TCP de service.

Le TCP_CNX doit être lancé à chaque tour de cycle automate pour gérer les connexions.

Remarque

Une connexion établie sera fermée :

- Suite à une demande par l'application PL7 d'annuler un transfert en cours.
- Sur une fermeture de la connexion par l'application distante.
- Suite à un accès par une machine non autorisée (ouverture temporaire pour contrôle).

La détection d'une erreur du caractère "fin de message" sur la réception d'un message par le coupleur n'entraîne pas la fermeture automatique de la connexion par le coupleur.

Par contre, la connexion sera marquée comme désynchronisée, prenant les caractéristiques suivantes :

- Toutes les données reçues sur une connexion désynchronisée sont perdues.
- Les nouvelles demandes de réception sur une connexion désynchronisée sont immédiatement retournées (sans aucune donnée) avec une erreur connexion désynchronisée.
- Les émissions sont possibles sur une connexion désynchronisée.

Toutes les connexions seront fermées :

- Suite à un RESET logiciel (Reset UC, téléchargement d'appli PL7).
- Suite à un RESET matériel du coupleur (coupure secteur, changement d'adresse sur bornier, ...).
- Suite à un RESET du DFB TCP_CNx.

Remarque

Les connexions rompues brutalement pour lesquelles l'automate ne serait pas prévenu (arrêt brutal de l'application distante, coupure secteur du distant, fermeture de connexion par le distant avec câble débranché...), peuvent donner lieu à un blocage de la communication sur le port attaché à cette connexion. En effet une demande de réception peut rester bloquée pendant un maximum de deux heures (temps du time-out de KEEP ALIVE). De plus toute nouvelle connexion sur ce port durant ce laps de temps sera refusée par l'automate.

Pour remédier à ce problème, il est conseillé à l'applicatif automate d'émettre régulièrement un message d'entretien de la connexion (le message de longueur 0 sera accepté par le DFB TCP_SEND).

3.2-2 Transfert de messages

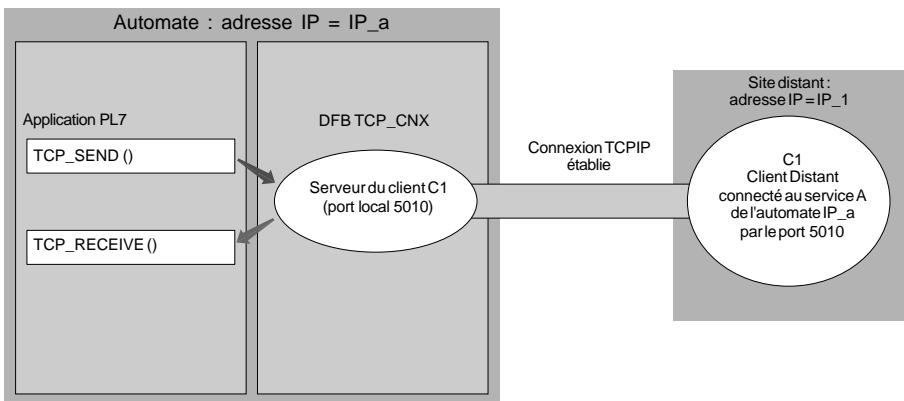
Description du service de transfert

Le service de transfert permet d'échanger des **messages** (suite de N octets) sur une connexion établie entre un client distant et l'application PL7 de l'automate sur un port local donné.

Dans les demandes de transfert, deux possibilités sont offertes à l'application PL7 pour identifier le client distant :

1. Par le seul paramètre : numéro de port local de l'automate (paramètre suffisant pour identifier une connexion TCP du coupleur).
Le transfert sera effectué si une connexion est établie sur ce port, quelle que soit la machine distante connectée à ce port.
2. Par le couple de paramètres : numéro de port local et adresse IP de la machine distante.
Le transfert sera effectué si une connexion est établie sur ce port avec la machine distante spécifiée par l'application PL7.

Transfert de message d'application à application



Les services de transfert de messages offerts à l'application PL7 de l'automate, sont les suivants :

- Demande d'émission vers un client distant d'un message dont la taille maximale est de 8 K octets.
Une émission de message restera active tant que les deux conditions suivantes sont réalisées :
 - le tampon d'émission du point connexion émetteur (automate) est plein,
 - le tampon de réception du point de connexion destinataire est plein,c'est à dire si le destinataire ne consomme pas les données sur la connexion.
- Demande de réception d'un client distant d'un message dont la taille maximale est de 8 K octets. L'attente de réception est active tant que la connexion est présente et qu'aucun message complet n'est reçu sur cette connexion. Si nécessaire, il sera du ressort de l'application PL7 d'utiliser un timer (câblé sur le signal RST du DFB) pour annuler la demande en cours au bout d'un délai.

Si la connexion TCP n'est pas établie avec le client distant au moment de la demande de transfert de données (émission ou réception), cette dernière sera immédiatement refusée avec une indication d'erreur : **connexion non établie**.

De même, si la connexion est fermée par le client distant ou perdue, toutes les opérations en cours sur cette connexion seront annulées.

Transfert multiple sur un même port

L'application PL7 ne peut pas activer en parallèle plusieurs demandes de même nature (émission ou réception) sur un même port, sans attendre le compte rendu de la demande précédente.

Cependant, il n'y a aucune corrélation temporelle entre les émissions et les réceptions. Pour une même connexion, le canal en émission et le canal en réception sont indépendants et donc l'application PL7 peut demander simultanément une émission et une réception sur le même port.

Format des messages échangés entre le coupleur et l'application distante

Une caractéristique importante de la communication sur une connexion TCP est l'aspect continu de l'information ("flots d'octets") ; le récepteur n'a pas la possibilité de retrouver la structure d'un message sans information et protocole particulier établi entre l'émetteur et le destinataire.

Il a été choisi de structurer les messages échangés par le format suivant :

2 octets	N octets	
Longueur message = N (format réseau)	Données du message utilisateur (N-1) octets	Caractère Fin de Message

Au niveau de l'automate, les deux champs supplémentaires (longueur message et caractère fin de message) seront ajoutés (émission) ou supprimés (réception) par les DFBs (TCP_SEND ou TCP_RECEIVE).

Longueur message :

- Ce champ spécifie le nombre total d'octets du message utile, constitué par les données utilisateur plus le caractère optionnel de fin de message.
- Il est défini comme une valeur signée sur 16 bits au format réseau. Il permet de coder des valeurs de 0 à 32 767 ($2^{16} - 1$).

Caractère "Fin de message" :

- Ce champ sur 1 octet est un caractère optionnel, pour marquer la fin du message.
- Cette marque de fin est une information redondante avec le paramètre "longueur de message" qui est suffisant pour structurer le flot de données en messages. Son utilité est uniquement de contrôler en réception sur le coupleur la cohérence du message et d'éviter la propagation d'une erreur provoquée par une application distante.

Procédure d'émission sur la connexion TCP

La procédure d'émission sur une connexion TCP établie est déclenchée par le DFB TCP_SEND de l'application PL7.

La procédure d'émission du DFB TCP_SEND est effectuée par bloc de 240 octets vers le coupleur TSX ETY 110 WS/5102 qui transmet immédiatement le bloc vers le distant. Les champs "longueur" et "caractère fin de message" (si configuré) sont automatiquement transmis par le DFB TCP_SEND.

Le DFB signale la fin de l'échange au moyen de son bit d'activité.

Procédure de réception du coupleur sur la connexion TCP

Les données reçues sur une connexion TCP établie sont stockées dans les tampons du coupleur TSX ETY 110 WS/5102. Les données reçues sur la connexion ne seront lues que si un DFB TCP_RECEIVE est actif.

La taille du tampon de réception est au maximum de 4096 octets. Ceci implique que lors de l'émission d'un message de 8 K par un distant, la moitié des données sont stockées dans ses tampons d'émission tant que l'application PL7 n'a pas reçue la première moitié des données.

La procédure de réception du TCP_RECEIVE est la suivante :

- Attente/Réception* des 2 octets indiquant la longueur N du message à recevoir.
- Réception* des N octets du message et du caractère fin de message si l'option caractère fin de message est dans la configuration. Le contrôler et le supprimer.
- Le DFB signale la fin de l'échange au moyen de son bit d'activité.

Remarque

Si une erreur "caractère fin de message" est détectée, le tampon de réception de l'application PL7 contient le message reçu qui peut être analysé.

S'il manque des caractères par rapport à la longueur spécifiée dans l'en-tête du message (erreur applicative de l'émetteur), la réception reste en attente infinie sur les octets manquants. Elle sera débloquée sur réception du message suivant, mais avec une détection d'erreur "caractère fin de message" (si le caractère fin de message ne coïncide pas avec une donnée utile du message).

La réception est alors désynchronisée.

La détection d'erreur de fin de message n'est pas forcément immédiate avec le premier message suivant (dans le cas où il y a coïncidence du caractère fin de message avec une donnée du message suivant, il peut passer plusieurs messages avant détection).

* L'opération Attente/Réception sur la connexion permet de recevoir également l'indication de fermeture de la connexion par la machine distante.

Protocole d'application à application

Aucune contrainte n'est imposée sur le protocole d'échange d'une connexion TCP établie entre le client distant et l'application automate. Les échanges peuvent être définis comme :

- Requête/réponse à l'initiative de l'application client.
- Requête/réponse à l'initiative de l'application automate.
- Messages non sollicités à l'initiative de l'application client.
- Messages non sollicités à l'initiative de l'application automate.

Caractéristiques d'utilisation du transfert par l'application PL7

Nombre de ports TCP

Le coupleur est dimensionné pour 16 ports TCP et donc 16 connexions.

Tampon utilisateur associé au DFB

Lors des échanges UC/coupleur, le DFB d'émission ou de réception utilise directement le tampon donné par l'utilisateur (il n'y a pas de recopie des données utilisateur dans un tampon système). Donc, tant que le DFB est actif (bit ACTIVITY à 1), le programme PL7 ne doit en aucun cas modifier le contenu du tampon associé à ce DFB.

4.1 Les points communs des DFBs

Les blocs fonction utilisent les services de messagerie implémentés dans le processeur PREMIUM. De ce fait, **tous les blocs fonction de communication peuvent s'exécuter sur plusieurs cycles automatés.**

Du fait de l'architecture d'échange coupleur/UC, les messages en émission (8 K octets) ou en réception (8 K octets) seront échangés entre le coupleur et l'application PL7 par datagrammes de 256 octets contenant des octets d'informations de contrôle de la segmentation et 240 octets de données utiles.

L'échange UC/coupleur d'un datagramme est effectué par cycle automate.

4.1-1 L'appel entretenu

L'exécution des blocs fonction est entretenue de manière explicite. L'utilisateur doit programmer l'entretien du DFB tant que son bit ACTIVITY est actif.

Il est strictement interdit d'exécuter plus d'une fois la même instance de DFB dans un même cycle automate.

L'entretien des DFBs est obligatoire du fait que tous les blocs fonction de communication s'exécutent sur plusieurs cycles automate.

Dès que le DFB est appelé, celui-ci est lancé de nouveau à chaque tour de cycle automate tant que son bit d'activité est égal à 1.

4.1-2 La table de gestion

Il est nécessaire que l'application PL7 fournisse aux DFB une table de gestion de 67 mots qui permet à chaque DFB de connaître l'état des connexions en cours. Chaque entrée de la table est constituée par des mots de 16 bits et comporte l'adresse IP, le numéro de port et le status de la connexion avec un distant.

Cette table est définie dans l'automate en zone mémoire MW et à la structure suivante :

%MWi	MODULE	Numéro de module
%MWi + 1	NP	Nombre de ports d'écoute
%MWi + 2	EOM	Caractère de fin de message
%MWi + 3	IP 1	Adresse IP de machine distante n° 1
%MWi + 5	P 1	Numéro de port local n° 1
%MWi + 6	Status 1	Status de la connexion n° 1
%MWi + 7	IP 2	
%MWi + 62	Status 15	
%MWi + 63	IP 16	Adresse IP de machine distante n° 16
%MWi + 65	P 16	Numéro de port local n° 16
%MWi + 66	Status 16	Status de la connexion n° 16

La structure de cette table est donnée à titre d'information. Cette table est entièrement gérée par les DFBs et ne doit pas être modifiée sans activer l'entrée RESET du DFB de gestion des connexions.

• Données internes

MODULE - Numéro de module

Numéro de l'emplacement physique du coupleur TSX ETY 110 WS/5102 dans le rack.

NP - Nombre de ports d'écoute

Ce paramètre, compris entre 1 et 16 inclus, définit les NP premiers éléments de la table des numéros de port locaux à utiliser.

EOM - Caractère Fin de Message

Ce paramètre définit la présence et la valeur d'un caractère "fin de Message" à utiliser dans les échanges de messages. Il est mis à jour par le DFB TCP_CNx lors de son premier appel ou après un RESET avec la valeur paramétrée par l'application automate :

- EOM = 0 : pas de caractère fin de message.
- EOM de 0x01 à 0xFF : valeur du caractère fin de message :
 - à ajouter lors d'une émission,
 - à contrôler et supprimer lors d'une réception.

IP i - Adresse IP du client connecté au port local n° i

Ce paramètre est mis à jour par le DFB TCP_CNx lorsqu'un client est connecté (valeur 0 si pas de connexion sur ce port). Il est ensuite utilisé par les DFB TCP_SEND et TCP_RECEIVE pour contrôler le paramètre d'entrée "Adresse IP" lorsque celui-ci n'est pas nul.

P i - Numéro de port local n° i

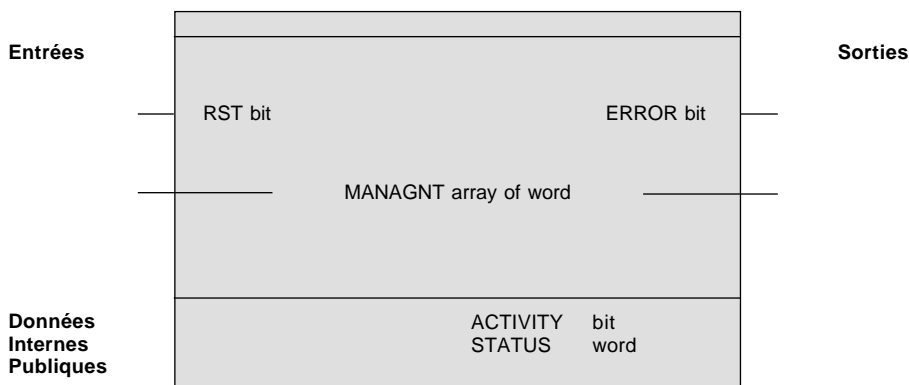
Tableau de 16 ports locaux de service à écouter (valeur signée \geq à 5010). Pour une configuration donnée, seuls les NP premiers numéros seront utilisés.

Status i - status de connexion n° i

Ce paramètre permet de donner des informations sur la connexion associée :

- Bits 0..3 : Numéro de socket de service compris entre 0 et 15.
- Bit 8 : 1 \bar{O} Connexion établie 0 \bar{O} pas de connexion.
- Bit 9 : 1 \bar{O} Emission en cours.
- Bit 10 : 1 \bar{O} Réception en cours.
- Bit 11 : 1 \bar{O} Erreur sur dernière émission.
- Bit 12 : 1 \bar{O} Erreur sur dernière réception.

4.1-3 Les bits RST et ACTIVITY



Tous les blocs de fonction TCP utilisent la messagerie et s'exécutent sur plusieurs cycles automate. C'est pourquoi l'utilisateur doit pouvoir interrompre l'exécution d'un bloc (en attente d'une réponse à un message) ; il doit également être en mesure de savoir si une exécution est en cours ou non, afin d'entretenir ou non son activité.

Le bit ACTIVITY

Les blocs fonction TCP ont une variable de type BIT qui indique leur activité ; c'est à dire, s'ils doivent ou non être entretenus.

L'action RST

Les blocs fonction TCP disposent d'une entrée RST de type BIT qui permet d'interrompre leur exécution sur la mise à 1 de ce bit, provoquant les actions suivantes (si la connexion est ouverte) :

- Les messages éventuellement en cours (émission et réception) sont perdus pour la connexion.
- La connexion est fermée.

4.1-4 Le bit ERROR - mot STATUS

Ce bit de sortie est mis à l'état 1 si le DFB ne peut pas fonctionner correctement. Le mot STATUS indique alors le type d'erreur.

• Mot STATUS

Bit	DFB	Signification	Action corrective
Bit 0 = 1	C R S	Le module n'est pas un coupleur ETY 110	Vérifier la valeur utilisée pour le paramètre d'entrée DEVICE de la configuration. Vérifier la configuration matérielle de l'automate.
Bit 1 = 1	C R S	Erreur syntaxe	RESET du DFB TCP_CNX. Vérifier les paramètres et redémarrer.
Bit 2 = 1	R S	Le port demandé est occupé par une machine dont l'adresse IP est différente de celle spécifiée dans le paramètre d'entrée (DEST pour émission, FRM pour réception).	Vérifier la configuration des clients distants. Renseigner le paramètre DEST ou ou FRM pour sélectionner correctement le client distant.
Bit 3 = 1	R S	Port local incorrect. Le numéro de port n'appartient pas à la liste des ports locaux donnée lors de la configuration du coupleur	Vérifier la liste des ports d'écoute du DFB TCP_CNX.
Bit 4 = 1	R S	Longueur erronée	En émission vérifier que le paramètre SIZE est inférieur à 8192. En réception une entête de message a été reçue avec le champ longueur > à 8192.
Bit 5 = 1	R S	La connexion n'est pas ouverte	Vérifier que le client est présent.
Bit 6 = 1	R S	Perte de connexion en cours de transfert	Vérifier le bon fonctionnement du client.
Bit 7 à 9		Non utilisés	
Bit 10 = 1	R	Connexion désynchronisée en réception. Un message a été reçu avec une erreur de caractère "fin de message"	Analyser le message dans le tampon de réception pour connaître l'origine de l'erreur.

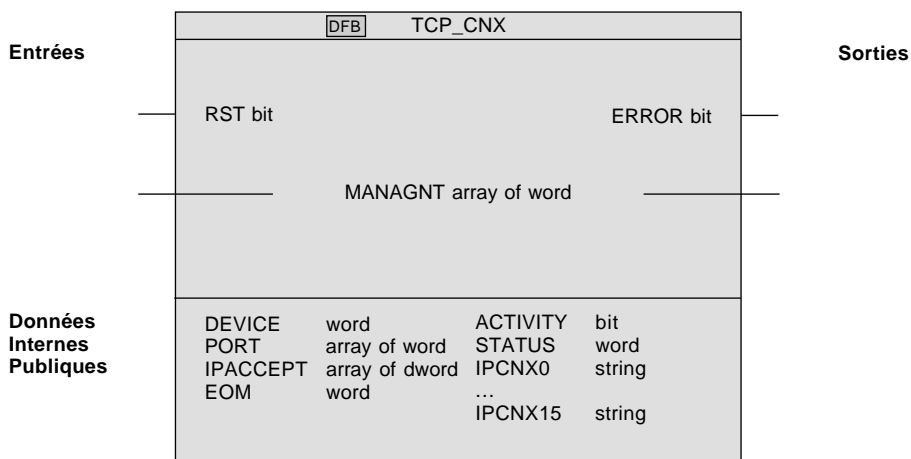
Bit	DFB	Signification	Action corrective
Bit 11 = 1	R	Erreur Caractère Fin de message Une erreur de caractère "fin de message" a été détectée sur ce message	Analyser le message dans le tampon de réception pour connaître l'origine de l'erreur. Faire un RESET du DFB.
Bit 12 = 1	R	Message tronqué. La taille du tampon de réception est trop petite pour recevoir complètement le message envoyé par le client distant	Prévoir un tampon plus grand. Vérifier la taille maximale des messages échangés entre applications sur cette connexion.
Bit 13 = 1	R	Non utilisé	
Bit 14 = 1	R S	Traitement interrompu. Le bloc fonction a été interrompu pendant son exécution par une action RST ou une reprise à chaud ou à froid.	Relancer la connexion côté client. Le serveur se met automatiquement de nouveau à l'écoute.
Bit 15 = 1		Non utilisé	

5.1 Présentation

Le DFB TCP_CNX est chargé de gérer les connexions avec les clients distants ainsi que les modes de marche de ces connexions. Il est renseigné des ruptures de connexions au moyen du status des mots de gestion par un bit positionné par une instance de DFB TCP_SEND ou TCP_RECEIVE.

Une seule instance du DFB TCP_CNX doit exister pour un module TSX ETY 110 WS/5102 donné.

5.1-1 Caractéristiques



Paramètres d'entrées

Paramètres	Type	Description
RST	Bit	La mise à l'état 1 de cette entrée : <ul style="list-style-type: none">- interrompt les échanges en cours,- provoque la fermeture de toutes les connexions.

Paramètres d'entrées/sorties

Paramètres	Type	Description
MANAGNT	Tableau de mot	Table de gestion commun à tous les DFB.

Paramètres de sorties

Paramètres	Type	Description
ERROR	Bit	Ce bit de sortie est mis à l'état 1 si le DFB ne peut pas fonctionner correctement. Le mot STATUS indique alors le type d'erreur qui s'est produite.

Données internes publiques

Paramètres	Type	Variables Ecriture/Lecture	Description
DEVICE	Mot	E	Numéro de module Numéro de l'emplacement physique du coupleur TSX ETY 110 WS/5102 dans le rack.
PORT	Tableau de mots	E	Numéro de port local n° i Tableau de 16 ports locaux de service à écouter (valeur signée ≥ 5010). La valeur 0 indique une valeur non significative.
IPACCEPT	Tableau de doubles mots	E	Adresse IP de la machine distante n° i Tableau d'adresses IP sur 4 octets de 8 machines distantes autorisées à se connecter à l'automate. La valeur 0 indique une valeur non significative.
EOM	Octet	E	Caractère "fin de Message" Ce paramètre définit la présence et la valeur d'un caractère "Fin de Message" à utiliser dans les échanges de messages : - EOM = 0 : pas de caractère fin de message, - EOM de 0x01 à 0xFF : valeur du caractère fin de message : -ajouté par le coupleur lors d'une émission - contrôlé et supprimé par le coupleur lors d'une réception.
IPCNXi	Chaîne de caractères	L	Adresse IP de la machine distante n° i Chaîne de caractère en lecture seule uniquement destinée à la maintenance et indiquant sous le format aaa.bbb.ccc.ddd l'adresse IP de la machine distante connectée au port d'indice i dans la configuration. La valeur 0.0.0.0 indique qu'aucune machine n'est connectée à ce port.
ACTIVITY	Bit	L	Ce bit de sortie est à un quand le DFB est en cours et a besoin d'être entretenu. Il est mis à l'état 0 sur reprise à chaud ou à froid et sur RESET du DFB.
STATUS	Mot	L	Ce mot est significatif uniquement si le bit de sortie ERROR (échange erroné) est à l'état 1. Il indique le code d'erreur produite lors de l'échange (Chaque bit du mot positionné à 1 indique une erreur). Se reporter au chapitre "Points commun des DFBs".

5.1-2 Fonctionnement

Le DFB TCP_CNX doit être appelé à chaque tour de cycle automate pour une gestion permanente des ports TCP.

L'écoute des ports configurés débute si les bits RST et ACTIVITY sont à 0. Ensuite le bit ACTIVITY reste positionné à 1 tant que l'application laisse le bit RST à 0.

Si le dialogue avec le coupleur n'est pas correct, le bit de sortie ERROR (échange erroné) est mis à l'état 1.

A tout instant, la mise à 1 du bit RST (entrée prioritaire) permet d'interrompre toutes les connexions en attente ou en cours. Le bit ACTIVITY (échange terminé) est mis à l'état 0 et le bit ERROR (échange erroné) est mis à l'état 1. Le mot status <nom de l'instance du TCP_CNX>.STATUS indique le type d'erreur.

Lors d'une reprise à froid ou à chaud le DFB TCP_CNX se remet automatiquement en écoute sur les ports configurés.

5.1-3 Exemple de programmation - Connexion sur 1 seul port TCP

• Données utilisées

%MW0 :	Table de gestion
%KD200 : H'5A000001'	Adresse IP (90.0.0.1) de la machine distante autorisée à se connecter
%KW300 : 5010	Port de service
%KW0 : 2	Emplacement physique du coupleur ETY110
%KW1 : 15	Caractère fin de message
%M0	Bit d'erreur
%MW100 :	Variable de travail pour indexer les tables
TCP_COX	Nom de l'instance du DFB TCP_CNX

```
! < EMISSION DE MESSAGE >
IF NOT TCP_COX.ACTIVITY THEN

    (* Initialisation de la configuration *)
    TCP_COX.DEVICE := %KW0 ;

    (* mise à 0 préalable de la table *)
    FOR %MW100 :=0 TO 15 DO
        TCP_COX.PORT[%MW100]:=0 ;
    END_FOR;
    (* Liste de ports d'écoute *)
    TCP_COX.PORT[0]:= %KW300;

    (* mise à 0 préalable de la table *)
    FOR %MW100 :=0 TO 7 DO
        TCP_COX.IPACCEPT[%MW100]:=0;
    END_FOR;
    (* Liste des adresses IP autorisées à se connecter *)
    TCP_COX.IPACCEPT[0]:= %KD200;

    (* Caractère de fin de message *)
    TCP_COX.EOM:= %KW1;

    (* lancement du DFB *)
    TCP_COX (0, %MW0:67, %M0 );
ELSE
    (* TRAITEMENT COMPTE-RENDU message précédent *)
    IF %M0 THEN JUMP %L3;
    (* entretien du DFB *)
    TCP_COX(0, %MW0:67 , %M0 );
    END_IF;
END_IF;
```

Note :

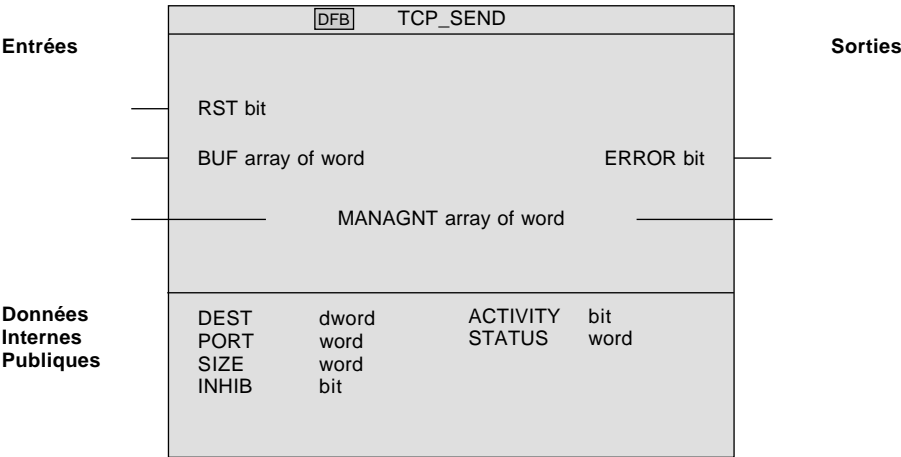
Passage des paramètres au DFB TCP_CNX

< nom de l'instance du DFB TCP_CNX> (RST, MANAGNT, ERROR)

6.1 Présentation

Le DFB TCP_SEND permet d'envoyer un message de données vers une application "cliente" distante, au travers d'une connexion TCPIP. La taille maximale du message à émettre est de 8 K octets.

6.1-1 Caractéristiques



Paramètres d'entrées

Paramètres	Type	Description
RST	Bit	La mise à l'état 1 de ce bit : - interrompt l'échange en cours (si le bit ACTIVITY était à 1) - provoque la fermeture de la connexion (si elle est ouverte) - positionne le bit d'information ACTIVITY à l'état 0 et le bit de sortie ERROR à l'état 1. Le code de l'erreur est alors contenu dans le mot STATUS.
BUF	tableau de mots	Cette variable définit l'adresse du premier mot %MWi du message à émettre.

Paramètres d'entrées/sorties

Paramètres	Type	Description
MANAGNT	Tableau de mot	Table de gestion commun à tous les DFB.

Paramètres de sorties

Paramètres	Type	Description
ERROR	Bit	Ce bit de sortie est mis à l'état 1 si l'échange ne s'est pas terminé correctement. Le mot STATUS indique alors le type d'erreur qui s'est produite.

Données internes publiques

Paramètres	Type	Variable Ecriture/Lecture	Description
DEST	Double mot	E	Cette variable définit l'adresse IP de la machine distante sur laquelle s'exécute le client connecté au port local PORT. Par défaut DEST = 0, le message est envoyé vers l'unique client distant connecté au port local de service de l'automate.
PORT	Mot	E	Cette variable définit le numéro de port local auquel le client distant est connecté.
SIZE	Mot	E	Cette variable définit la taille en octets du message à émettre. De 0 à 8192. Ne tient pas compte du caractère de fin.
ACTIVITY	Bit	L	Ce bit est à 1 quand un échange est en cours. Il est mis à l'état 0 lorsque l'échange est terminé.
INHIB	Bit	L	Ce bit permet d'inhiber la signalisation des erreurs : le bit de sortie ERROR et le mot STATUS restent toujours à 0 (l'exécution du bloc n'est pas interrompu).
STATUS	Mot	L	Ce mot est significatif uniquement si le bit de sortie ERROR (échange erroné) est à l'état 1. Il indique le code d'erreur produite lors de l'échange (Chaque bit du mot positionné à 1 indique une erreur). Se reporter au chapitre "Les points communs des DFBs".

6.1-2 Fonctionnement

Le transfert de données est déclenché lors de l'appel du DFB TCP_SEND si le bit RST est à l'état 0 et si le bit interne ACTIVITY est à l'état 0 (aucun échange en cours). Au cours de l'échange, le bit ACTIVITY est positionné à 1. En fin d'émission, le bit ACTIVITY est mis à l'état 0. De plus, si l'échange n'est pas correct, le bit de sortie ERROR (échange erroné) est mis à l'état 1.

A tout instant, la mise à 1 du bit RST (entrée prioritaire) permet d'interrompre l'échange en cours. Le bit ACTIVITY (échange terminé) est mis à l'état 0 et le bit ERROR (échange erroné) est mis à l'état 1. Le mot status <nom de l'instance du TCP_SEND>.STATUS indique le type d'erreur de l'échange. Si la connexion est ouverte, elle se ferme et le coupleur repasse en attente de connexion rentrante. C'est au rétablissement de la connexion sur ce port TCP de service, que l'émission des messages peut reprendre. Cette reconnexion est gérée par le DFB TCP_CNX.

Sur le déclenchement d'un émission de message, le traitement est le suivant :

- Contrôle des paramètres d'entrée : adresse IP de la machine distante, numéro de port local.
- Recherche de la connexion ouverte avec le client distant demandé.
- Emission du message à émettre par bloc de 240 octets (en ajoutant le caractère "fin de message" si l'option est demandée par configuration) sur la connexion TCP.

6.1-3 Exemple de programmation

• Données utilisées

%MW0 :	Tableau de gestion
%KD200 : H'5A000001'	Adresse IP (90.0.0.1) de la machine distante (sur 4 octets)
%KW300 : 5010	Port de service
%W200 : 400	Taille en mots du message à émettre
%W300 à W700	Message à émettre
%M0	Bit d'erreur
TCP_EMIS	Nom de l'instance du DFB TCP_SEND

```
! < EMISSION DE MESSAGE >
IF NOT TCP_EMIS.ACTIVITY THEN
    ( * TRAITEMENT COMPTE-RENDU message précédent *)
    IF %M0 THEN JUMP %L3;

    ( * Emission message suivant *)
    TCP_EMIS.PORT:=%KW300;
    TCP_EMIS.DEST:=%KD200;
    TCP_EMIS.SIZE:=%MW200*2;
    ( * lancement du DFB le premier paramètre qui représente RST à 0 *)
    TCP_EMIS(0, %MW300:400, %MW0:67, %M0);
    END_IF;
ELSE
    ( * entretien du DFB le premier paramètre qui représente RST à 0 *)
    TCP_EMIS(0, %MW300:400, %MW0:67, %M0);
END_IF;
```

Note :

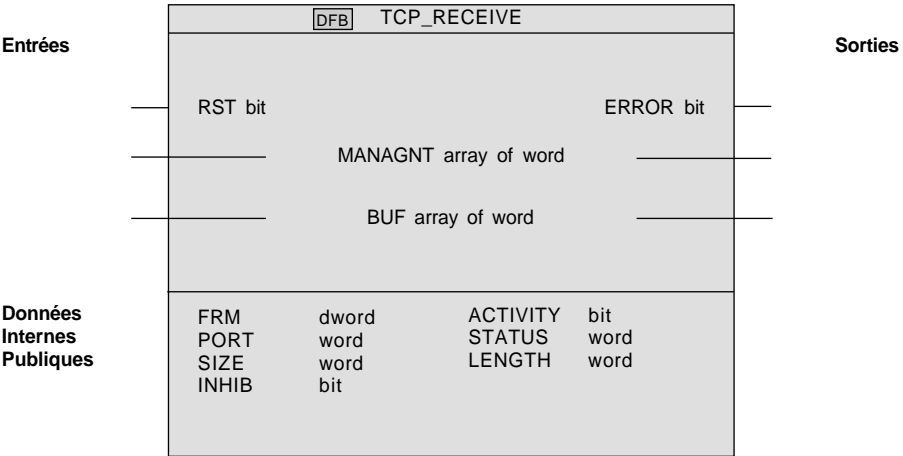
Passage des paramètres au DFB

<nom de l'instance du TCP_SEND> (RST, BUF, MANAGNT, ERROR)

7.1 Présentation

Le DFB TCP_RECEIVE permet de recevoir un message d'une application "cliente" distante, au travers d'une connexion TCPIP. La taille du message à recevoir peut atteindre 8 K octets.

7.1-1 Caractéristiques



Paramètres d'entrées

Paramètres	Type	Description
RST	Bit	La mise à l'état 1 de ce bit : - interrompt l'échange en cours (si le bit ACTIVITY était à 1) - provoque la fermeture de la connexion (si elle est ouverte) - positionne la variable ACTIVITY à l'état 0 et le bit de sortie ERROR à l'état 1. Le code de l'erreur est alors contenu dans le mot STATUS.

Paramètres d'entrées/sorties

Paramètres	Type	Description
BUF	Tableau de mot	Ce paramètre d'entrée et de sortie définit l'adresse du premier mot %MWi du tampon pour recevoir le message.
MANAGNT	Tableau de mot	Tableau de gestion commun à tous les DFB.

Paramètres de sorties

Paramètres	Type	Description
ERROR	Bit	Ce bit de sortie est mis à l'état 1 par le DFB si l'échange ne s'est pas terminé correctement. Le mot STATUS indique alors le type d'erreur qui s'est produite.

Données internes publiques

Paramètres	Type	Variable Ecriture/Lecture	Description
FRM	Double mot	E	Cette variable définit l'adresse IP de la machine distante sur laquelle s'exécute le client connecté au port local PORT. Par défaut FRM = 0, le DFB se met en attente de message envoyé par l'unique client distant connecté au port local de service de l'automate.
PORT	Mot	E	Cette variable définit le numéro de port local auquel le client distant est connecté.
SIZE	Mot	E	Cette variable définit la taille en octets du tampon pour recevoir le message. De 0 à 8192.
ACTIVITY	Bit	L	Cette variable est à l'état 0 par le DFB lorsque l'échange est terminé. Si le bit de sortie ERROR (échange erroné) est à l'état 0, la variable ACTIVITY indique que le message a été correctement reçu. Par contre, si le bit ERROR est à l'état 1, la variable ACTIVITY indique que l'échange est terminé mais erroné
INHIB	Bit	E	Ce bit permet d'inhiber la signalisation des erreurs : le bit de sortie ERROR et le mot STATUS restent toujours à 0 (l'exécution du bloc n'est pas interrompu)
STATUS	Mot	L	Ce mot est significatif uniquement si le bit de sortie ERROR (échange erroné) est à l'état 1. Il indique le code d'erreur produite lors de l'échange (chaque bit du mot positionné à 1 indique une erreur). Se reporter au chapitre "Les points commun des DFBs"
LENGTH	Mot	L	Cette variable contient le nombre d'octets reçus si le bit de sortie ERROR (échange erroné) est à l'état 0.

7.1-2 Fonctionnement

Le transfert de données est déclenché lors de l'appel du DFB TCP_RECEIVE si l'entrée RST est à l'état 0 et si le bit ACTIVITY est à l'état 0 (aucun échange en cours). Au cours de l'échange, le bit ACTIVITY est positionné à 1. En fin de réception, le bit ACTIVITY est mis à l'état 0. De plus, si l'échange n'est pas correct, le bit de sortie ERROR (échange erroné) est mis à l'état 1.

A tout instant, la mise à 1 de l'entrée RST (entrée prioritaire) permet d'interrompre l'échange en cours. Le bit ACTIVITY (échange terminé) est mis à l'état 0 et le bit ERROR (échange erroné) est mis à l'état 1. Le mot status <nom de l'instance du TCP_RECEIVE>.STATUS indique le type d'erreur de l'échange. Si la connexion est ouverte, elle se ferme et le coupleur repasse en attente de connexion rentrante. Si un ou plusieurs messages sont dans les tampons de réception du coupleur, ils sont perdus.

Sur le déclenchement d'une réception de message, le traitement est le suivant :

- Contrôle des paramètres d'entrée : identification de la machine distante, numéro de port local,...
- Recherche de la connexion ouverte avec l'application distante demandé.
- Mise en file de l'EF de réception pour attente de message sur cette connexion.

Dès qu'un DFB est actif pour une connexion donnée, le coupleur se met en attente de réception de données sur cette connexion, les deux premiers octets reçus donnant la longueur du message à recevoir. Les données reçues sont alors transférées dans les mots de l'automate par trames de 240 octets.

Dès que le message complet est recopié dans l'automate :

1. Le DFB vérifie le caractère fin de message (si l'option a été configurée).
2. Les paramètres STATUS, LENGHT du DFB sont renseignés et la variable ACTIVITY et le bit ERROR sont alors positionnés pour l'application PL7.

7.1-3 Exemple de programmation

• Données utilisées

%MW0 :	Tableau de gestion
%KD200 : H'5A000001'	Adresse IP (90.0.0.1) de la machine distante (sur 4 octets)
%KW300 : 5010	Port de service
%W200 : 400	Taille en mots du tampon de réception
%W300 à W700	Tampon de réception
%M0	Bit d'erreur
TCP_RECEP	Nom de l'instance du DFB TCP_RECEIVE

```
! < EMISSION DE MESSAGE >
IF NOT TCP_RECEP.ACTIVITY THEN
    (* TRAITEMENT COMPTE-RENDU message précédent *)
    IF %M0 THEN JUMP %L3;

    (* Emission message suivant *)
    TCP_RECEP.PORT:=%KW300;
    TCP_RECEP.FRM:=%KD200;
    TCP_RECEP.SIZE:=%MW200*2;
    (* lancement du DFB *)
    TCP_RECEP(0, %MW0:67, %MW300:400, %M0);
    END_IF;
ELSE
    (* entretien du DFB le premier paramètre qui représente RST à 0 *)
    TCP_RECEP(0, %MW0:67, %MW300:400, %M0);
    END_IF;
```

Note :

Passage des paramètres au DFB

<nom de l'instance du TCP_RECEIVE> (RST, MANAGNT, BUF, ERROR)

8.1 Points intervenants dans les performances d'une communication TCPIP-OPEN

Les performances d'une communication TCPIP-OPEN dépendent des points suivants :

- Du temps de cycle automate
Celui-ci tient compte du temps de traitement process et du temps de traitement réservé à la communication TCPIP-OPEN.
- Du temps de cycle périodique
Celui-ci doit prendre en compte le temps de traitement du coupleur TSX ETY 110 WS/5102 afin que la réponse à la demande faite au coupleur arrive au cycle automate suivant.
- Du temps de traitement du coupleur TSX ETY 110 WS/5102. Celui-ci est inférieur à 30 ms pour 4 ports TCP configurés.
- Du nombre maximum d'EF lancé dans un même temps de cycle automate. Celui-ci est lié au type de processeur.

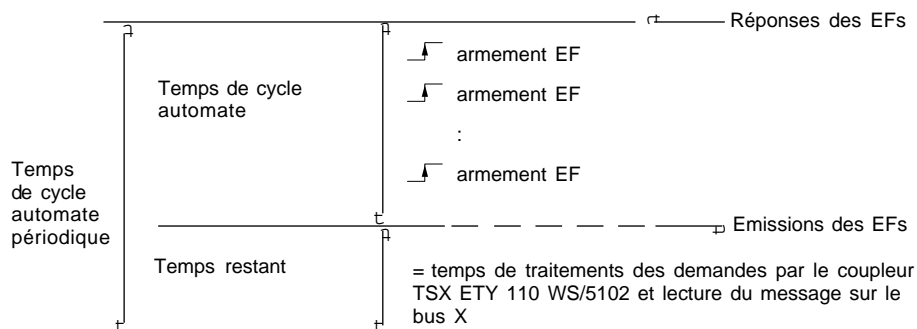
Remarque

Les meilleures performances sont obtenues avec un flux de communication régulier en émission et en réception. Ce fonctionnement est optimal lorsqu'un DFB peut envoyer un EF à chaque cycle automate.

L'EF étant envoyé en fin de cycle automate, il est impératif que sa réponse arrive au début de la période suivante.

Pour cela, le temps de lecture du message sur le bus X et le temps de traitement du coupleur ETY110WS/5102 doivent être inférieur au temps restant entre la fin du cycle automate et le début de la période suivante.

8.2 Schéma d'un traitement d'EF dans un cycle automate



Note :

Les EFs TCPIP-OPEN sont utilisés par les DFBs, TCP_CNx, TCP_SEND, TCP_RECEIVE. Se reporter au document "OPEN TCP for TSX Premium" concernant la structure et le fonctionnement des EFs TCPIP-OPEN.

8.3 Mesures de performances

Ces mesures de performances ont été réalisées entre une application automate serveur TCPIP-OPEN sur Premium TSXP57352 et une application cliente TCPIP-OPEN sur PC sous Windows NT.

Temps de cycle périodique de l'automate	50 ms	70 ms	80 ms	90 ms
Temps de réponse d'un message de 8 K octets émis sur 4 ports de service TCP	3,7 s	3,33 s	3 s	3,75 s
Temps de réponse d'un message de 8 K octets émis sur 2 ports de service TCP	3 s			

9.1 Différences avec les OFB PL7-3

L'offre est similaire sur la gamme Série 7 avec ETH110. Les différences sont principalement :

- **Performances** : Le coupleur ETH110 ne supporte que le profil de communication TCP/IP_OPEN. Les performances avec le coupleur TSX ETY 110 WS/5102 seront fonction de l'utilisation des autres fonctionnalités du coupleur (Messagerie UNITE, Messagerie MODBUS, HTTP, SNMP, FTP).

L'exécution de plusieurs DFB **simultanés** (dans un même cycle automate) destinés à des échanges de même nature avec un même distant (identifié par le numéro de port et/ou l'adresse IP) n'est pas possible car il n'est pas possible de le gérer dans l'architecture TCOPEN.

- **Modes de marches** : La **gestion des connexions** est réalisée par le coupleur dans l'offre ETH110 alors qu'elle **est à la charge de l'applicatif automate** dans la démarche actuelle avec TCOPEN.
- **Configuration** : Sur ETH110, la configuration comporte les paramètres du coupleur, la liste des ports utilisés ainsi que la liste des adresses IP distant autorisées. Dans l'offre TCOPEN, les paramètres du coupleur sont configurés par PL7 et gérés par le système. La liste des ports utilisés et des adresses IP autorisées est gérée par le DFB (ou EF) TCP_CNX.
- **Programmation PL7** : La méthode de programmation est différente : Les **DFBs doivent être** entretenus par l'applicatif, alors que les OFB de ETH110 sont entretenus automatiquement.
La **fonction Echo n'existe pas** dans l'architecture TCOPEN.
- **Vision externe réseau et distant** : L'entité message est connue du coupleur ETH110 alors qu'elle n'est connue que de l'automate dans l'offre TCOPEN. Ce qui implique pour le cas de l'émission d'un message, la **fragmentation en unités de 240 octets sur le réseau**. L'**interruption d'émission** par le Premium d'un message en cours peut entraîner une perte de la fin du message.
Pour la réception, la vue réseau est identique au comportement ETH110.
- **Diagnostic** : Les outils de diagnostic utilisés pour les OFB de la série 7 (Applidiag) doivent être remplacés par des outils standards d'exploitation des DFBs intégrés dans le PL7.

9.2 Rappel sur la communication TCPIP

Se reporter aux ouvrages de type :

- Préparation au MCSE TCPIP, éditions S & SM.
- Préparation au MCSE TCPIP, éditions CAMPUSPRESS.
- TCPIP de Douglas Comer, éditions InterEditions.