RECAPITULATIF DES DOCUMENTS

Présentation générale		8 pages
Questionnement		31 pages
Annexes		
Module Processeur GMI_CPU10	A-1	10 pages
Schéma carte GMI_CPU10	A-1-2	1 pages
La carte GMI-RAM3	A-2	2 pages
La carte GMI-ETHER	A-3	4 pages
La carte GMI-SIO4	A-4	3 pages
Desc. Fonc. Des Systèmes	B-1	2 pages
Dossier Spécification	B-1-1	6 pages
Documentation OS9-68K	B-2	3 pages
The standard library OS-9	B-2-1	4 pages
Doc. Socket Unix	C-1	6 pages
Doc. inetd	C-2	2 pages
Prog. Windows BC++3.1	D-1	6 pages
Programmes Windows	D-2	18 pages
Windows Sockets	D-3	11 pages
Synoptique Général	G-1	1 pages

Module processeur GMI-CPU10

1) Table d'index.

1) Table d'index. 1
2) Caractéristiques. 1
3) Découpage mémoire 2
4) Désignation des connecteurs 2
5) Signal RESET et ABORT. 2
6) Interruptions 3
7) Configurateur EPROM/E2PROM. 3
8) Configurateur RAM. 4
9) Temps d'accès EPROMS 5
10) Configuration E, SYSCLK, BGRT, IACK. 5
11) Configuration test 6
12) Basculement EPROM/RAM. 6
13) Zone périphérique 6800 synchrone. 6
14) Zone périphérique 68000 asynchrone. 7
15) Watch-Dog 7
16) Interruptions niveau 7 7
17) Ports séries 8
18) Circuit horloge DS1287. 9
19) Arbitration du bus 9
20) Signaux bus G64-G96 10

2) Caractéristiques.

Processeur : 68030
coprocesseur Arithmétique : 68882
Horloge micro à 25 Mhz ou 33 Mhz
Bus d'adresses 32 bits .
Bus de données 32 bits non multiplexé avec le bus d'adresse (transfert de données : 8, 16 ou 32 bits).
Ressources internes : Accessibles en DMA par le bus.

Mémoire interne : RAM statique de 256 ko, 1 Mo ou 4 Mo (32 bits), mode asynchrone ou RAM statique rapide 128 ko ou 512 ko (32 bits), mode synchrone.
EPROM, 4 supports pour 32k x 8 à 1Mo x 8 (128k à 4 Moctets).
Extension RAM (32 bits) GMI-RAM5.4 (4 Mo), mode burst.
Extension RAM (32 bits) SEMI-RAM5.8 (8 Mo), mode burst
Mémoire externe : 2*11 Moctets adressables par le bus G96, cartes mémoires associées (16 bits)
- GMI-RAM3 (2 Mo)
- GMI-MEM3 (support RAM/EPROM)

Zone périphériques : Zone VPA de 2 koctets, pour périphériques externes synchrones et asynchrones.
Ports séries :
- 1 ligne synchrone/asynchrone RS232C,
- 1 ligne asynchrone RS232C,
- Transmissions FULL/HALF duplex,
- Vitesse et mode de transmission programmables,
- Niveau d'interruption 6,
- Interruptions vectorisées,
- Connecteurs de sortie 2x10 points,
- Compteur programmable 16 bits.

Chien de garde : Circuit de surveillance.
Horloge Calendrier : Sauvegarde.
Demande de DMA : Le protocole est établi par les signaux chain-in, chain-out et bus request.

Interruptions :

- 5 niveaux d'interruption sur le bus, sélection par niveau du vecteur ou de 1'autovecteur,
- 1 niveau non masquable, auto-vectorisé,
- 1 niveau utilisé pour les périphériques internes à la carte.
- La connaissance d ' interruption est effectuée grâce aux signaux chain-in, chain-out et Iack.

Interface bus G96 : Conforme aux spécifications du bus G96.
Système d'exploitation : OS9 (*) temps réel (Unix like).

3) Découpage mémoire

Organisation mémoire :

000000	**- 3FFFFF**	**Si bascule RAM/EPROM = 1 => RAM interne à la carte**
		Si bascule RAM/EPROM = 0 => EPROM interne à la carte (état au RESET)
400000	**- BFFFFF**	**Extension RAM interne 32 bits PIGGI_BACK**
C00000	**- F9FFFF**	**VMA**
FAFFFF	**- FA07FF**	**VPA externe synchrone (périphérique 6800)**
FA0800	**– FA0FFF**	**VPA externe asynchrone (redondant avec le VPA externe synchrone)**
FA1000	**– FAFFFF**	**Zone redondante**
FB0000	**- FB7FFF**	**VPA interne synchrone**
FB8000	**- FBFFFF**	**VPA interne asynchrone**
1000000	**- 13FFFFF**	**EPROM**
1400000	**- 17FFFFF**	**Si bascule RAM/EPROM = 0 => RAM interne à la carte (état au RESET)**
		Si bascule RAM/EPROM = 1 => EPROM interne à la carte
1800000	**- 1BFFFFF**	**VMA**
1C00000	**- 1EFFFFF**	**VMA (Réservé)**

Zones périphériques :

MC 68681	**FB8001**	**Registre du circuit**
Horloge	**FB0009**	**Registre d'adresse**
	FB000B	**Registre de données**
Bascule	**1FFFFFF**	**D0=0 => Eprom en 0**
		D0=1 => Ram en 0
Watch-dog	**FB001F**	**D0=1 => Déclenchement**
Interruption N°7	**FB0801**	**Registre d'interruptions niveau 7**

4) Désignation des connecteurs

DESIGNATION	FONCTION
P1	Connecteur bus G96
P2	Connecteur RS232C Port A
P3	Connecteur RS232C Port B
P4	Connecteur Reset
P5	Connecteur Extension RAM interne
P6	Connecteur Extension RAM interne
J1	Supprimé
J2	Configurateur Horloge E (1 MHz ou 2 MHz)
J3	Configurateur RAM INTERNE
J4	Configurateur signal E et SYCLK
J5	Configurateur fréquence SYCLK
J6	Configurateur BGRT ou BGRT/
J7	Configurateur temps d'accès RAM
J8	Configurateur temps d'accès EPROM
J9	Configurateur IACK ou IACK/
J10	Configurateur mémoire cache
J11	Configurateur Watch-dog
J12 à J16	Configurateur mode d'interruptions
J17	Configurateur signal ABORT
J18	Configurateur EPROM
J19	Configurateur EPROM/EEPROM
J20 à J22	Configurateurs EPROM
J23	Configurateur Modem/Terminal P2
J24	Configurateur EPROM
J25	Configurateur Modem/Terminal P3
J26	Alimentation connecteur P2, P3
J27	Configurateur test
J28	Configurateur MMU
J29, J30	Configurateurs RAM INTERNE

5) Signal RESET et ABORT.

Le signal RESET est activé par le bouton poussoir BP1 ou par un autre système utilisant les signaux du connecteur P4
Le signal ABORT est activé par le bouton poussoir extérieur et génère une interruption de niveau 7.

J17	Installé	pas de bouton ABORT sur P4
J17	Enlevé	utilisation d'un bouton ABORT sur P4

6) Interruptions

Les 7 niveaux d'interruptions sont utilisés. Un niveau est utilisé en interne, un niveau peut être utilisé en interne ou en externe, les cinq autres sont libres d'utilisation.

NIVEAU	GENERATION	TYPE
1	IRQ1 (IRQ1)/	Autovectorisé, J12 installé Vectorisé, J12 enlevé
2	IRQ2 (FIRQ)/	Autovectorisé, J13 installé Vectorisé, J13 enlevé
3	IRQ3/	Autovectorisé, J14 installé Vectorisé, J14 enlevé
4	IRQ4/	Autovectorisé, J15 installé Vectorisé, J15 enlevé
5	IRQ5/	Autovectorisé, J16 installé Vectorisé, J16 enlevé
6	DS1287 MC68681	Autovectorisé, Vectorisé,
7	NMI/ EXTPWF/ ABORT/	Autovectorisé,

7) Configurateur EPROM/E2PROM.

Le support EPROM est prévu pour recevoir une mémoire de 32, 64, 128, 256, 512 ou 1024 Koctets suivant la configuration des cavaliers sur J18 à J22 et J24

```
MEMOIRE                                 CAVALIERS

                    J18         J20         J21         J22         J24

                    o           o           o           o           o
27256                                       |
(32Ko)              o           o           o           o           o
                                                        |
                    o           o           o           o           o

                    o           o           o           o           o
27512                           |
(64Ko)              o           o           o           o           o
                                            |           |
                    o           o           o           o           o
                    o           o           o           o           o
271001                          |                                   |
(128Ko)             o           o           o           o           o
                    |                       |           |
                    o           o           o           o           o

                    o           o           o           o           o
272001                          |                       |           |
(256Ko)             o           o           o           o           o
                    |                       |
                    o           o           o           o           o

                    o           o           o           o           o
274001              |           |                       |           |
(512Ko)             o           o           o           o           o
                                            |
                    o           o           o           o           o

                    o           o           o           o           o
278001              |           |                       |
(1024Ko)            o           o           o           o           o
                                            |                       |
                    o           o           o           o           o
```

Les supports EPROM peuvent recevoir des E2 PROM 8kx8 ou 32kx8

8) Configurateur RAM.

Les mémoires RAM statiques utilisables sont de deux types :
a) Statiques standard (temps d'accès 70ns)
b) Statiques rapides (temps d'accès 25 ou 35 ns)

Chaque type a) ou b) peut équiper la carte GMI-CPU10 et les caractéristiques de la carte sont alors les suivantes :

Avec des RAM statiques standard :

GMI-CPU10 A-256
256 Ko SRAM 70ns 0 Wait state à 25 MHz
Fonctionnement en mode asynchrone ;
Soit un temps de cycle (écriture ou lecture) de 120ns

GMI-CPU10 B-256
256 Ko SRAM 70ns 1 Wait state à 33 MHz
Fonctionnement en mode asynchrone ;
Soit un temps de cycle (écriture ou lecture) de 120ns

GMI-CPU10 A-1Mo
1 Mo SRAM 70ns 0 Wait state à 25 MHz

GMI-CPU10 B-1Mo
1 Mo SRAM 70ns 1 Wait state à 33 MHz

GMI-CPU10 A-4Mo
4 Mo SRAM 70ns 0 Wait state à 25 MHz

Avec des RAM statiques rapides :

GMI-CPU10 A-R128
128 Ko SRAM 35ns 0 Wait state à 25 MHz
Fonctionnement en mode synchrone 2 cycles ;
Soit un temps de cycle (écriture ou lecture) de 80ns

GMI-CPU10 B-R128
128 Ko SRAM 25ns 0 Wait state à 33 MHz
Fonctionnement en mode synchrone 2 cycles ;
Soit un temps de cycle (écriture ou lecture) de 60ns

GMI-CPU10 A-R512
512 Ko SRAM 35ns 0 Wait state à 25 MHz
Fonctionnement en mode synchrone 2 cycles ;
Soit un temps de cycle (écriture ou lecture) de 80ns

GMI-CPU10 B-R512
512 Ko SRAM 25ns 0 Wait state à 33 MHz
Fonctionnement en mode synchrone 2 cycles ;
Soit un temps de cycle (écriture ou lecture) de 60ns

Le configurateur J7 sera positionné en fonction du type de SRAM utilisé (rapides ou standard)

Les configurateurs J3 et J29 sont ajustés en fonction du type de SRAM standard utilisé 32Kx8, 128Kx8 ou 512Kx8

```
                            o                                        o
RAM 32K        J3                              J29
                            o                                        o
                            |
                            o

                            o                                        o
RAM 128K       J3           |                  J29
                            o                                        o

                            o

                            o                                        o
RAM 512K       J3                              J29                   |
                            o                                        o

                            o
```

9) Temps d'accès EPROMS

J8 permet de modifier le temps qu'il y a entre les signaux LDS, UDS et DTACK, c'est à dire d'ajuster le temps du cycle du processeur, en fonction du temps d'accès des mémoires utilisées. Le temps de cycle du processeur est variable puisque 2 fréquences d'horloge (25 et 33 MHz) sont proposées.

```
              5    4    3    2    1
              o    o    o    o    o
J8
EPROM         o    o    o    o    o
              6    7    8    9    10
```

Le tableau ci-dessous indique la configuration optimale en fonction du temps d'accès EPROM et de la fréquence d'horloge du 68030.

	Temps d'accès max EPROM		Etat d'attente (Wait state)
J8	25 MHz	33 MHz	
1---10	250 ns	180 ns	5
2---9	200 ns	150 ns	4
3---8	150 ns	120 ns	3
4---7	120 ns	85 ns	2
5---6	85 ns	55 ns	1
pas de cavalier	55ns	35 ns	0

10) Configuration E, SYSCLK, BGRT, IACK.

Le cavalier J2 permet le choix de deux fréquences pour le signal E utilisé par les périphériques synchrones. Ces deux fréquences dépendent de celle du 68030.

Fréquence micro		Cavalier J2
25 MHz	33 MHz	
781 KHz	1 MHz	enlevé
1,56 MHz	2 MHz	installé

J4	1-2	E et SYCLK	Toujours actif sur le bus
J4	2-3	E et SYCLK	En tri-state pour DMA
J5	1-2	SYCLK=CPUCK	
J5	2-3	SYCLK=CPUCK/2	
J6	1-2	BGRT/	Actif à l'état bas
J6	2-3	BGRT	Actif à l'état haut

J9	Installé	IACK	Actif à l'état haut
J9	Enlevé	IACK/	Actif à l'état bas

Mémoire cache

La mémoire cache du 68030 peut être validée ou invalidée par J10

J10	Installé	Mémoire cache INVALIDEE
J10	Enlevé	Mémoire cache VALIDEE

MMU

La MMU du 68030 peut être validée ou invalidée par J28

J28	Installé	MMU INVALIDEE
J28	Enlevé	MMU VALIDEE

Watch-dog

Le cavalier J11 permet de rendre actif l'action du Watch-dog sur le reset ou de l'invalider indépendamment de la programmation.

J11	Installé	Watch-dog ACTIF
J11	Enlevé	Watch-dog INACTIF

11) Configuration test

J27	Installé	Fonctionnement normal
J27	Enlevé	Configuration de test

12) Basculement EPROM/RAM.

Le 68xxx ayant besoin des vecteurs « Reset » et « Système Stack Pointer » en 0 pour l'implantation de OS9, l'EPROM située en FC0000 est également présente en 0.
Pour disposer de RAM en 0, on procède de la manière suivante :
Le programme qui démarre en FC0000 doit commencer par le vecteur Reset
A l'entrée du programme, basculer l'Eprom en RAM à l'adresse 0. Pour cela, écrire « 1 » à l'adresse 1FFFFFF (D0=1)
Recopier ensuite les vecteurs initialisés de FC0000 en 0

Il est possible de travailler avec la mémoire située au delà des vecteurs initialisés en 0.

13) Zone périphérique 6800 synchrone.

Les périphériques 8 bits sont adessés par le VPA qui décode un champs de 1 Koctets. Le 68000 travaillant sur 16 bits, adresse les octets bas ou hauts d'une carte. Ceci implique que les périphériques G64/G96 6800 sont adressés dans un champ de 2 Koctets, et que ce champ est redondant de FA à FB.

Pour des raisons de compatibilité, il faut adresser ces périphériques de FA0000 à FA07FF.

Les adresses de base des cartes sont toujours impaires, et les registres sont adressés, par incréments de 2.

Exemple : Carte GMI-SIO1 en $100 du VPA

	6809	68000
ACIA1CR	VPA+ $100	VPA+ $201
ACIA1DR	VPA+ $101	VPA+ $203
ACIA2CR	VPA+ $102	VPA+ $205
ACIA2DR	VPA+ $103	VPA+ $207
ACIA3CR	VPA+ $104	VPA+ $209
ACIA3DR	VPA+ $105	VPA+ $20B
ACIA4CR	VPA+ $106	VPA+ $20D
ACIA4DR	VPA+ $107	VPA+ $20F
	VPA=$FC00	VPA=$FC0000

14) Zone périphérique 68000 asynchrone.

Cette zone décodée par un Pal permet de travailler avec des cartes périphériques, conçues pour les périphériques 68000. Adresser un périphérique dans cette zone permet au 68000 d'attendre le signal DTACK.

Attention : Cette zone est redondante avec la zone VPA synchrone.

Exemple : Carte GMI-INP1 adressée en VPA+ 0 :
Les registre de cette carte sont également présents à VPA+$800. La lecture en VPA synchrone + 0 permet d'accéder à la carte, sans que celle-ci génère de DTACK (périphériques synchrone). La lecture en VPA asynchrone + 0 gébèrera une erreur, car la carte ne génère pas de DTACK.

15) Watch-Dog

Ce circuit de surveillance, lorsqu'il est activé, permet de générer un RESET, s'il n'est pas entretenu. La durée est réglée à environ 0,5 seconde.

Exemple : t0 $01 dans le registre : armement du Watch-Dog

t1 $00 dans le registre : entretien du Watch-Dog
$01 dans le registre : entretien du Watch-Dog

t1+DT $00 dans le registre : entretien du Watch-Dog
$01 dans le registre : entretien du Watch-Dog

Si (t1-t0)>0,5 seconde ou DT>0,5 seconde, alors RESET du système.

16) Interruptions niveau 7

Le registre est accessible à l'adresse $FB0801

D3	D2	D1	D0
Signal PWF du Bus	Déclenchement du Watch-Dog	Bouton poussoir ABORT	Ligne NMI du Bus

Après le Reset

Les interruptions sont masquées mais la lecture du registre indique l'état des lignes avant le Reset.

Exemple : 1011 indique que le Watch-Dog a été déclenché (cause probable du Reset s'il a été activé)
1111 indique qu'aucune interruption n'a été enregistrée.

Reset logiciel et masquage des interruptions

Ecrire 0000 dans le registre
Lecture 1111 état des lignes sans interruptions

Démasquage des interruptions

Ecrire 1111 dans le registre
4 sources d'interruptions sont démasquées en même temps.

Reconnaissance d'interruption

Lecture xxxx si le bit est à 1, pas d'interruption
Si le bit est à 0, l'interruption correspondante a été déclenchée.

Acquittement de l'interruption

Ecriture	1110	acquittement IT NMI
	1101	acquittement IT ABORT
	1011	acquittement Watch-Dog
	0111	acquittement IT PWF
	0000	interdit, reset et masque toutes les interruptions

En écriture, les valeurs 0000 et 1111 sont privilégiées. L'une reset et masque les interruptions, l'autre démasque les interruptions.

17) Ports séries

Description

Deux lignes synchrones/asynchrones RS232 sont disponibles sur le module GMI-CPU10. Les signaux sont accessibles sur les connecteurs P2 et P3 qui réalisent l'interface direct par câble plat avec les prises Cannon 25 broches normalisées. Les broches à wrapper J23 et J25 permettent de configurer le mode de transmission. Les transmission sont gérées par le circuit MC68681.

En mode synchrone, le montage du circuit ne permet pas de générer les fréquences d'horloge à l'émission et à la réception. Par contre, deux entrées d'horloges permettent de synchroniser le circuit avec un étage de sortie, qui fournit l'horloge à l'émission et à la réception, pour le canal B uniquement.

Pour la réalisation d'adaptateurs de communications l'alimentation peut être disponible sur les connecteurs P2 et P3 en installant le cavalier J26.

Signaux

CONNECTEUR J23 J25	SIGNAL	DESCRIPTION
8	RxD	Réception des données
6	TxD	Emission des données
3	RTS	Demande pour émettre
13	CTS	Prêt à émettre
11	RxC	Horloge réception
10	TxC	Horloge transmission
14	+12V	Pullup 1K au +12V

Configuration Modem/Terminal

Ces configurations sont obtenues en réalisant les straps indiqués ci-dessous sur les connecteurs J23 et J25. Les connexions entre P2/P3 et le connecteur Cannon 25 points sont réalisés par câbles plats.

Liaison avec un terminal ou une imprimante, protocole géré par le signal DTR, mode asynchrone.

```
CANNON 25 points                         20 points P2/P3      J23/J25          ACIA
DTR     20-----------------------------  14     -------------------1o   o14---   +12V
CTS     5------------------------------  9      -------------------2o   o------  CTS
RTS     4------------------------------  7      --------------------o   o------  RTS
        8------------------------------  15     --------------------o   o------  RxC
        16-----------------------------  6      --------------------o   o------  TxC
RxD     3------------------------------  5      --------------------o   o------  TxD
TxD     2------------------------------  3      -------------------7o   o8----   RxD
0V      7------------------------------  1-13   ---------   0V
0V      1------------------------------  1
        18-------------------------------------10--------------------------------------o J24 o------    +5V
```

Liaison avec un terminal ou une imprimante, protocole logiciel XON-XOFF, mode asynchrone.

CANNON 25 points		20 points P2/P3	J23/J25		ACIA
DTR	20	14	1o	o14	+12V
CTS	5	9	2o	o	CTS
RTS	4	7	o	o	RTS
	8	15	o	o	RxC
	16	6	o	o	TxC
RxD	3	5	o	o	TxD
TxD	2	3	7o	o8	RxD
0V	7	1-13	0V		
0V	1	1			
	18	10	o J24 o		+5V

Vitesse de transmission /protocole

La vitesse et le protocole (7 ou 8 bits, avec ou sans parité, etc…) sont programmables par logiciel.

18) Circuit horloge DS1287.

Description

Le circuit horloge et calendrier permet à l'utilisateur de développer des applications temps réel.
Ce circuit occupe dans l'espace VPA interne synchrone, deux adresses

Registres

Le circuit horloge occupe les adresses FB0009 et FB000B.
L'accès aux registres du circuit est réalisé en deux temps :

1- Ecriture à l'adresse FB0009 du numéro du registre de travail
2- Lecture ou écriture à l'adresse FB000B du registre dont l'adresse a été donnée lors de la manipulation précédente.

Circuit de surveillance "Bus Time Out"

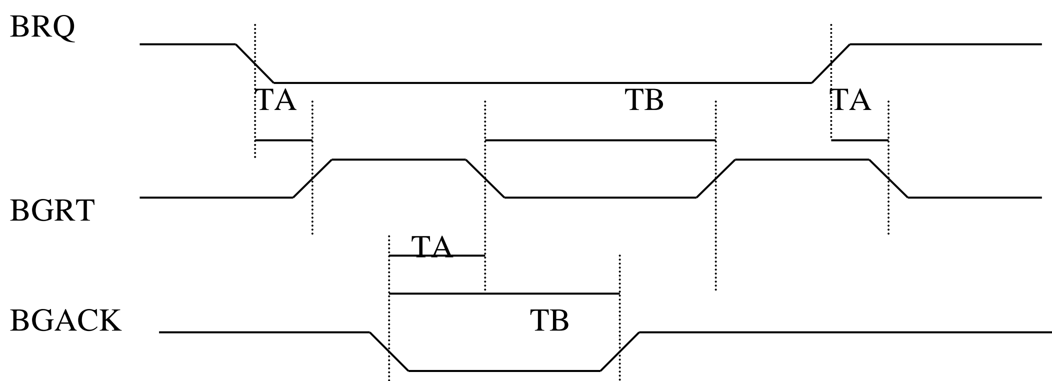
Ce circuit permet de ne pas bloquer le processeur, si celui-ci va adresser une case mémoire inconnue. Un cycle "Erreur bus" est déclenché, si le 68030 ne reçoit pas un des signaux DTACK ou VPA

FREQUENCE	25 MHz	33 MHz
TIME OUT	5 µs	3,8 µs

ARBITRATION DU BUS DAISY CHAIN

19) Arbitration du bus .

Un périphérique peut demander le Bus G64/G96 par l'intermédiaire de la ligne BRQ (Bus request). En réponse, BGRT (Bus grant) et BGACK (bus grant acknowledge) sont affirmés. Le Daisy Chain est ensuite utilisé pour fixer les priorités



T = 125 ns

TA = 1,5-3 T
TB = min 1,5 T

Daisy Chain

Le Daisy Chain est utilisé pour fixer la priorité entre plusieurs demandeurs, soit en reconnaissance d'interruption, soit en DMA. La propagation du signal a lieu entre modules adjacents, par les lignes chain-in et chain-out.

20) Signaux bus G64-G96

PIN	ROW C		ROW B		ROW A	
1	*	GND	*	GND	*	GND
2	*	A16	*	A8	*	A0
3	*	A17	*	A9	*	A1
4	*	A18	*	A10	*	A2
5	*	A19	*	A11	*	A3
6	*	A20	*	A12	*	A4
7	*	A21	*	A13	*	A5
8	*	A22	*	A14	*	A6
9	*	A23	*	A15	*	A7
10		Réservé	*	BRQ/	*	BGRT/ ou BGRT
11		Réservé	*	DS1/	*	DS0/
12	*	Réservé	*	BGACK/ ou BBUSY/	*	HALT/
13		GND	*	E	*	SYCLK
14		Réservé	*	RES/	*	VPA/
15	*	Réservé	*	NMI (IRQ7)/	*	RDY ou DTACK/
16	*	IRQ3/	*	IRQ1 (IRQ)/	*	VMA/
17		IRQ5/	*	IRQ2 (FIRQ)/	*	R et W/
18		VED/	*	IACK/ ou IACK	*	IRQ4/
19	*	GND	*	D12/	*	D8/
20		P5/	*	D13/	*	D9/
21		P4/	*	D14/	*	D10/
22		P3/	*	D15/	*	D11/
23		P2/	*	D4/	*	D0/
24		P1/	*	D5/	*	D1/
25		P0/	*	D6/	*	D2/
26	*	Réservé	*	D7/	*	D3/
27	*	SYSFAIL/	*	BERR/		Page/
28		ARBCLK/	*	Chain-in	*	Chain-out
29		Réservé	*	5 V BAT	*	PFW/
30		Réservé	*	"- 12 V	*	"+ 12 V
31	*	"+5 V	*	"+5 V	*	"+5 V
32	*	GND	*	GND	*	GND

- * : Signal utilisé par le module
- / : Signal actif à l'état bas
- Signaux :
  /DS1 = /UDS
  /DS0 = /LDS

ANNEXE GMI-RAM3

Carte RAM

Carte jumelée à la carte CPU équipée de :
3 PALs
4 Modules de RAM 4 Mo x 9

Description

Le module GMI-RAM6 est une carte mémoire dynamique d'une capacité de 16, 32, 64, ou 128 Mo avec le transfert de données en 32 bits sur la carte GMI-CPU10

Un contrôle de parité est inclus sur la carte. Ce système permet de générer une erreur en cas de mauvais fonctionnement de cette dernière.

Un cavalier permet le choix ou non du contrôle de parité.

Caractéristiques

Transfert de données — 32 bits en mode synchrone
Types de mémoires — Barrettes SIMM
4 Mo x 9 (minimum 4 barrettes) ou 16 Mo x 9 (min 4 barrettes) 60, 70 ou 80 ns
Capacité mémoire — 16 Mo (4 barrettes 4Mo x 9)
32 Mo (8 barrettes 4Mo x 9)
64 Mo (4 barrettes 16Mo x 9)
128 Mo (8 barrettes 16Mo x 9)

Désignation des cavaliers configurateurs et connecteurs

DESIGNATION	FONCTION
J1	Sélection type de barrettes mémoires
J2, J3	Parité
J4, J5	Capacité mémoire utilisée
P1	Interface bus G64/G96
P2,P3	Connecteurs de liaison avec carte GMI-CPU10

Installation des barrettes mémoires:

Carte équipée de 16 Mo de mémoire (60, 70 ou 80 ns) à 25 MHz
Soit 4 barrettes SIMM de 4Mo x 9 installées
Cavalier J1 position 2-3
Cavalier J4 installé
Cavalier J5 installé
Cavalier J2 et J3 enlevés

Carte équipée de 32 Mo de mémoire (60, 70 ou 80 ns) à 25 MHz
Soit 8 barrettes SIMM de 4Mo x 9 installées
Cavalier J1 position 2-3
Cavalier J4 installé
Cavalier J5 enlevé
Cavalier J2 et J3 enlevés

Carte équipée de 64 Mo de mémoire (60, 70 ou 80 ns) à 25 MHz
Soit 4 barrettes SIMM de 16Mo x 9 installées
Cavalier J1 position 1-2
Cavalier J4 enlevé
Cavalier J5 installé
Cavalier J2 et J3 enlevés

Carte équipée de 128 Mo de mémoire (60, 70 ou 80 ns) à 25 MHz
Soit 8 barrettes SIMM de 16Mo x 9 installées

Cavalier J1 position 1-2
Cavalier J4 enlevé
Cavalier J5 enlevé
Cavalier J2 et J3 enlevés

Parité

Si l'on souhaite un fonctionnement avec parité: installer les cavaliers J2 et J3. En cas de mauvaise parité, la carte mémoire génèrera un bus erreur.

Le nombre de cycles sans parité est : 4 + 2 + 2 + 2
Le nombre de cycles avec parité est : 4 + 3 + 3 + 3
Pour une fréquence µP de 25 MHz

GMI-ETHER

Description

La carte GMI-ETHER est une carte de communication Ethernet avec sortie AUI

Caractéristiques

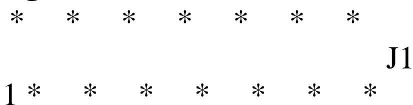
Contrôleur	AM7990
Connexion Ethernet	AUI
Interface BUS	Conforme au bus G96, adressage asynchrone, zone VPA, interruptions autovectorisées.

Configurateurs

Désignation des configurateurs

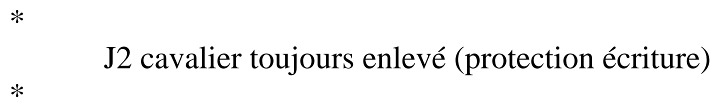
DESIGNATION	FONCTION
J1	Adressage de la carte
J2	Programmation de l'EEPROM
J3	Sélection des interruptions

Adressage de la carte

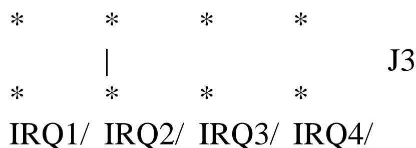


La carte occupe 16 octets dans la zone VPA asynchrone
Configuration de livraison : VPA asynchrone + $700

Programmation de l'EEPROM



Configuration des interruptions



Configuration sur IRQ2/

Organisation des registres

00-01	R/W	Registre données AM7990
02-03	R/W	Registre d'adresse AM7990
04	/W	Mise à 0 de la clock de l'EEPROM
05	/W	Mise à 1 de la clock de l'EEPROM
06	/W	Mise à 0 de la data de l'EEPROM
06	R	Lecture de la data de l'EEPROM
07	/W	Mise à 1 de la data de l'EEPROM
08-09	R/W	Data buffer tournant de 2048 octets avec auto-incrémentation
0A	R/W	N.U.
0B	W	Offset du buffer tournant A0-A7, mise à 0 de A8 et A9
0C-0D	R/W	Data buffer tournant sans auto-incrémentation
0E	R/W	N.U.
0F	W	Sélection de la page. Mise à 0 de A0 à A9

Logiciel associé

Présentation

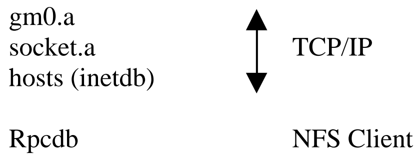
La carte est fournie avec le logiciel TCP/IP et NFS Client

Configuration matérielle

La configuration de livraison du matériel correspond à la configuration de livraison logiciel.
Adresse de la carte : FA0F00 (VPA asynchrone + $700)
Interruption niveau 2 en autovectorisé

Configuration logiciel

Un certain nombre de fichiers sont à modifier pour pouvoir être intégrés dans votre environnement



gm0.a

gm0.a est le descripteur de la carte associé au driver am 7990_eth1

Six paramètres sont modifiables dans ce fichier
- adresse de la carte
- numéro de vecteur
- niveau d'interruption
- adresse ethernet
- adresse de diffusion
- adresse internet

Les trois premiers paramètres ne sont à modifier que si la carte n'est pas utilisée dans l'environnement prédéfini.

Template for ifdgen descriptor generation utility

```
Nam le0_enc1 device descriptor module
Use /dd/defs/oskdefs.d
SCOPE set 0
TypeLang set (Devic<<8)+0
Attr_Rev set (ReEnt<<8)+0
 Psect le0_enc1, TypeLang,Attr-Rev, 1,0,0
 dc.l    0x00FA0F00    port address
 dc.b    26            auto-vector trap assignment
 dc.b    2             IRQ hardware interrupt level
 dc.b    1             Exclusive polling table priority
 dc.b    Updat_device  mode capabilities
```

Adresse internet
Adresse de Broad cart

Ces deux paramètres sont impérativement à modifier pour la configuration de votre environnement.

```
 align
net_name dc.b "",0
 align
linkaddr dc.b 0,0,0,0,0,0                          adresse ethernet (plus utilisé)
 align
broadcast_addr dc.w 2 AF_INET
 dc.w 0 port
 dc.b 192,9,200,0 broadcast address (internet)     adresse de diffusion
```

```
br_addr_siz equ *-broadcast_addr
 ifgt 16-br_addr_siz          padd to at least 16 bytes
 rept 16-br_addr_siz
 dc.b 0
 endr
 endc
interface_addr dc.w 2 AF_INET address family
 dc.w 0 port
 dc.b 192,9,200,10 direct address (internet)             adresse internet
in_addr_siz equ *-interface_addr
```

socket.a

Deux paramètres sont configurables dans ce fichier
le nom de la station
le nom de ou des descripteurs à initialiser au lancement de TCP/IP

```
 dc.w 0
 dc.w 0
net_name dc.b "test", 0, 0, 0, 0, 0, 0, 0, 0, 0
inet_name dc.b "inet", 0
eth_name dc.b "af_ether", 0
af_unix_name dc.b "af_unix", 0
rip_name dc.b "ip", 0
udp_name dc.b "udp", 0
tcp_name dc.b "tcp", 0
device_names dc.b "/lo0 /gm0"…
 rept 64
 dc.b 0
 endr
NilMgr dc.b "sockman", 0 file manager
NilDrv dc.b "sockdrv", 0 device driver
 Ends
```

Après modification, lancer la commande "make" pour la mise à jour du fichier

Hosts

Le fichier hosts est le fichier d'association entre les noms de stations et les numéros internet
Il est entièrement dépendant du site.

Après modification, lancer la commande "idbgen" pour la mise à jour du fichier inetdb

Rpcdb

Pour générer le fichier Rpcdb, un fichier de procédure gen est fourni.
Pour la configuration, il suffit de modifier le nom de la station.

Nota: L'exportation des périphériques n'est valable que dans le cas où vous possédez NFS Serveur qui n'est pas fourni en standard.

```
-t
-nx
del .server-map
del nFsm*
Rpcdbgen Test-s-r …..     nom de la station
* expantfs /h0 ….         Juste pour NFS Serveur
```

Après modification du fichier, lancer la commande gen pour la misa à jour du fichier Rpcdb.

Localisation des fichiers en version 3.0

- gm0.a /h0/OS9/SRC/IO/INET/DESC
- socket.a /h0/OS9/SRC/IO/INET/SOCKDESC

- hosts /h0/OS9/SRC/IO/INET/ETC
- Rpcdb /h0/NFS/ETC

Lancement de TCP/IP

/h0/OS9/SRC/SYS/ EXTENSIONS/startinet

Logiciels spécifiques à la carte

- gm0 descripteur de la carte
- am 7990_eth1 driver de la carte
- gmstat statistique réseau

ANNEXE GMI-SIO4

Description

Carte communication "SERIE"
La carte GMI-SIO4 est équipée de quatre ports d'entrées/sorties série RS232.

Caractéristiques

4 lignes asynchrone	RS232C
E/S complémentaires	2 entrées, 6 sorties (TTL)
Transmission	FULL/HALF duplex
Vitesse	50 à 38400 bauds par voie
Sélection du module	Terminal, imprimante ou modem
Interruptions	Sélection par voie de la ligne

Désignation des connecteurs et configurateurs

DESIGNATION	FONCTION
J1	Configurateur d'adresse
J2	Configurateur d'interruptions
J3	Configurateur port liaison série P4
J4	Configurateur port liaison série P5
J5	Alimentation + 5 V connecteur P4
J6	Alimentation + 5 V connecteur P7
J7	Alimentation + 5 V connecteur P5
J8	Configurateur port liaison série P2
J9	Configurateur port liaison série P3
J10	Alimentation + 5 V connecteur P2
J11	Alimentation + 5 V connecteur P6
J12	Alimentation + 5 V connecteur P3
P1	Connecteur Bus G96
P2	Connecteur de sortie U9
P3	Connecteur de sortie U9
P4	Connecteur de sortie U8
P5	Connecteur de sortie U8
P6	Connecteur entrées/sorties TTL U9
P7	Connecteur entrées/sorties TTL U8

Configuration des adresses

La carte occupe 32 octets dans l'espace périphérique VPA de 1 Koctets en G64
En G96, elle occupe 64 octets mais n'utilise que les adresses impaires et les bits D0 à D7 du Bus.
Des cavaliers sur J1 permettent de définir l'adresse par rapport au VPA
Exemple: VPA = FA0801 (adresse de base)

Tous les cavaliers sont installés

```
*      *      *      *      *
|      |      |      |      |
*      *      *      *      *
A9     A8     A7     A6     A5
```

Sélection des interruptions

Le choix se fait par le configurateur J2
Exemple d'interruption sur IRQ4

```
*      *      *      *      *      *
              |
*      *      *      *      *      *
IRQ7   IRQ5   IRQ4   IRQ3   IRQ2   IRQ1
```

Adresses des ports séries

ADRESSE RELATIVE	REGISTRES DUART	CONNECTEURS
0	DUART2	P4, P5

16 $10	DUART1	P2, P3

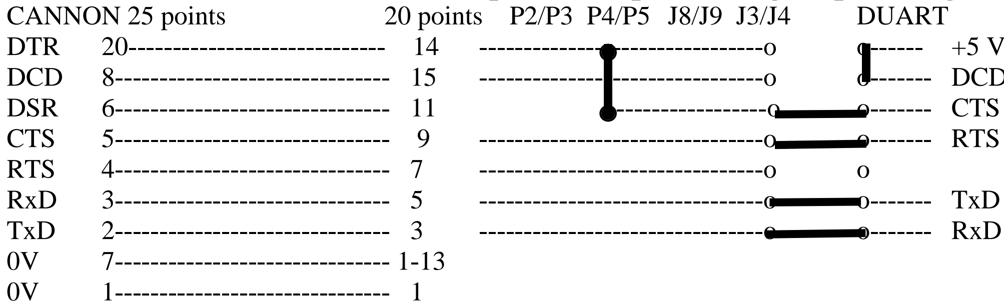
Signaux

CONFIGURATEUR J3/J4 J8/J9	SIGNAL	DESCRIPTION
8	RxD	Réception des données
6	TxD	Emission des données
4	RTS	Demande pour émettre
10,12	CTS	Prêt à émettre
13	DCD	Détection de la porteuse
14	+5 V	Pullup 3,3 K au +5 V

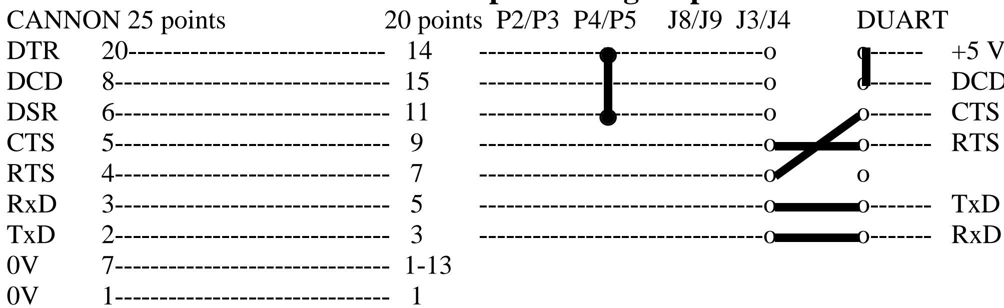
Configuration Modem/terminal

Ces configurations sont obtenues en installant des cavaliers sur J3, J4, J8, J9.

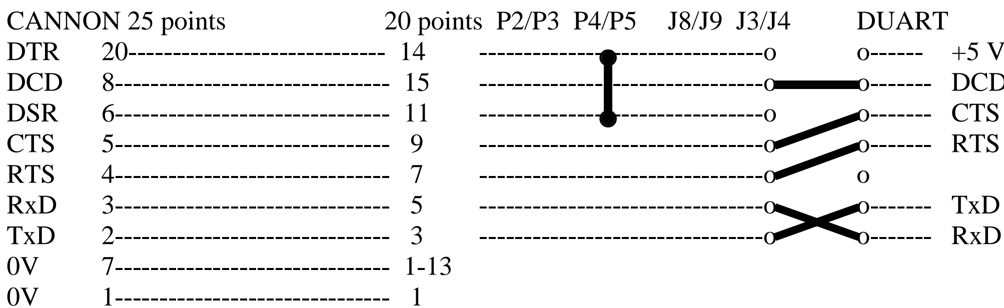
Liaison avec un terminal ou une imprimante, protocole géré par le signal DTR



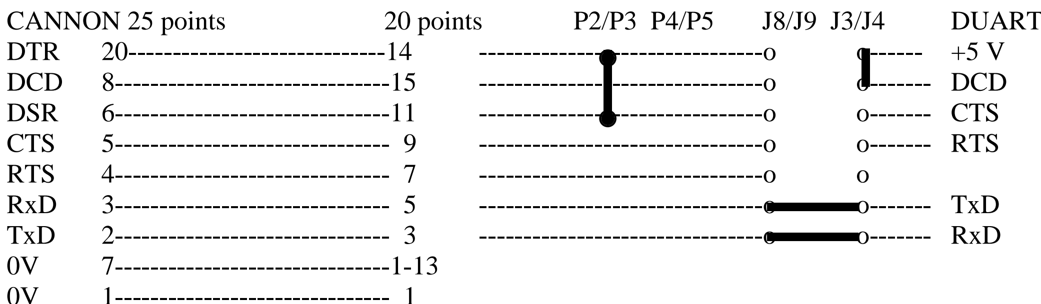
Liaison avec un terminal ou une imprimante géré par RTS



Liaison avec un modem



Liaison XON-XOFF



Configuration standard

Adresse carte $FA07C1 (aucun cavalier sur J1)
IRQ4 vectorisée (attention gestion CHAIN-IN CHAIN-OUT)
9600 bauds 8 bits sans parité, 1 bit de stop

Signaux bus G64-G96

PIN	ROW C			ROW B		ROW A
1		GND	*	GND	*	GND
2		A16	*	A8	*	A0
3		A17	*	A9	*	A1
4		A18	*	A10	*	A2
5		A19	*	A11	*	A3
6		A20	*	A12	*	A4
7		A21	*	A13	*	A5
8		A22	*	A14	*	A6
9		A23	*	A15	*	A7
10		Réservé		BRQ/		BGRT/ ou BGRT
11		Réservé		DS1/		DS0/
12		Réservé		BGACK/ ou BBUSY/		HALT/
13		GND	*	E		SYCLK
14		Réservé		RES/	*	VPA/
15		Réservé	*	NMI (IRQ7)/		RDY ou DTACK/
16		IRQ3/	*	IRQ1 (IRQ)/		VMA/
17		IRQ5/	*	IRQ2 (FIRQ)/	*	R et W/
18		VED/		IACK/ ou IACK		IRQ4/
19		GND		D12/		D8/
20		P5/		D13/		D9/
21		P4/		D14/		D10/
22		P3/		D15/		D11/
23		P2/	*	D4/	*	D0/
24		P1/	*	D5/	*	D1/
25		P0/	*	D6/	*	D2/
26		Réservé	*	D7/	*	D3/
27		SYSFAIL/		BERR/		Page/
28		ARBCLK/		Chain-in		Chain-out
29		Réservé		5 V BAT		PFW/
30		Réservé	*	"- 12 V	*	"+ 12 V
31		"+5 V	*	"+5 V	*	"+5 V
32		GND		GND	*	GND

* : Signal utilisé par le module
/ : Signal actif à l'état bas

DESCRIPTION FONCTIONNELLE DES SYSTEMES

LISTE DES INFORMATIONS ECHANGEES (liaison série) ENTRE L'AUTOMATE PRINCIPAL ET LE SYSTEME UC OS9

Nota :

- **les informations sont de type maintenues : pas d'information impulsionnelle.**
- **les chiffres situés dans les colonnes " Automate → UC" et " UC → Automate" précisent le nombre d'informations échangées pour la rubrique concernée.**

Informations	Automate → UC	UC → Automate
Interphone VL entrée sur pupitre 1		1
Interphone VL entrée sur pupitre 2		1
Interphone VL entrée sur pupitre 3		1
Interphone VL sortie sur pupitre 1		1
Interphone VL sortie sur pupitre 2		1
Interphone VL sortie sur pupitre 3		1
Interphone PL n°1 entrée sur pupitre 1		1
Interphone PL n°1 entrée sur pupitre 2		1
Interphone PL n°1 entrée sur pupitre 3		1
Interphone PL n°2 entrée sur pupitre 1		1
Interphone PL n°2 entrée sur pupitre 2		1
Interphone PL n°2 entrée sur pupitre 3		1
Interphone PL n°1 sortie sur pupitre 1		1
Interphone PL n°1 sortie sur pupitre 2		1
Interphone PL n°1 sortie sur pupitre 3		1
Interphone PL n°2 sortie sur pupitre 1		1
Interphone PL n°2 sortie sur pupitre 2		1
Interphone PL n°2 sortie sur pupitre 3		1
Sélection Auto barrière VL entrée	1	
Sélection Auto barrière VL sortie	1	
Ordre d'ouverture barrière VL entrée		1
Ordre d'ouverture barrière VL sortie		1
Défaut alimentation barrière VL entrée	1	
Défaut alimentation barrière VL sortie	1	
Sélection Auto 2 barrières PL entrée	1	
Sélection Auto 2 barrières PL sortie	1	
Défaut alimentation barrières Sas entrée (2)	1	
Défaut alimentation barrières Sas sortie (2)	1	
Défaut alimentation barrière PL n°1 entrée	1	
Défaut alimentation barrière PL n°2 entrée	1	
Défaut alimentation barrière PL n°1 sortie	1	
Défaut alimentation barrière PL n°2 sortie	1	
Ordre d'ouverture barrière PL n°1 entrée		1
Ordre d'ouverture barrière PL n°2 entrée		1
Ordre d'ouverture barrière PL n°1 sortie		1
Ordre d'ouverture barrière PL n°2 sortie		1
Alarme onduleur	1	
Défaut onduleur	1	
Lancement enregistrement magnétoscope		1
Prise en compte commande portail Engins de Parc par St Cassien		1
Demande d'ouverture portail Engins de parc par St Cassien		1
Demande de fermeture portail Engins de parc par St Cassien		1
Commande locale portail SNCF Nord	1	
A.U. portail SNCF Nord	1	

Palpeurs portail SNCF Nord	1	
Fdc portail ouvert SNCF Nord	1	
Fdc portail fermé SNCF Nord	1	
Véhicule devant cellules portail SNCF Nord	1	
Défaut manœuvre portail SNCF Nord	1	
Défaut dialogue avec automate SNCF Nord	1	
Informations portail SNCF Centre (idem SNCF Nord)	8	
Informations portail SNCF Sud (idem SNCF Nord) et ECM	16	
Informations portail SNCF Engins de Parc (idem SNCF Nord)	8	
Informations portail de réinjection (idem SNCF Nord sauf défaut dialogue automate)	7	
Informations portail Entrées terminal (idem SNCF Nord sauf défaut dialogue automate)	7	
Informations portail Entrées terminal (idem SNCF Nord sauf défaut dialogue automate)	7	
Défaut dialogue automate local portails E/S	1	
Total :	**76**	**28**

EXEMPLE D'ENTREES / SORTIES TRAITEES AU NIVEAU D'UN AUTOMATE LOCAL : AUTOMATE DEDIE AU PORTAIL SNCF NORD

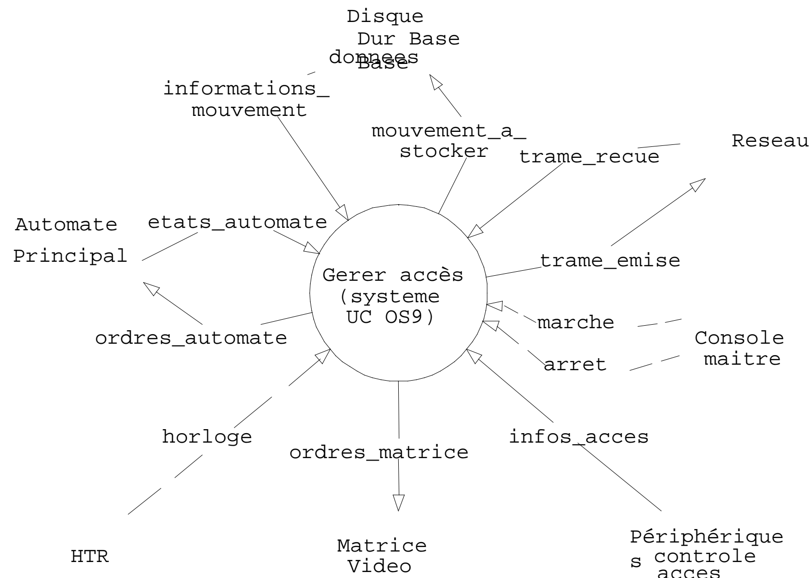
Informations	**Entrée T.O.R.**	**Sortie T.O.R.**
Commutateur commande Locale / Distante du portail	1	
Bouton Poussoir ouverture locale du portail	1	
Bouton Poussoir fermeture locale du portail	1	
Défaut palpeurs portail	1	
Fdc portail ouvert	1	
Fdc portail fermé	1	
Bouton Poussoir réarmement local sur défaut manœuvre du portail	1	
A.U. local	1	
Véhicule devant cellules portail	1	
Ouverture portail		1
Fermeture portail		1
Gyrophare clignotant		1
Arrêt distant Mourepiane	1	

EXEMPLE D'ENTREES / SORTIES TRAITEES AU NIVEAU D'UN MODULE MTCA : MODULE DEDIE A LA VOIE PL N°1

Informations	**Entrée T.O.R.**	**Sortie T.O.R.**
Présence véhicule au niveau de la voie PL n°1	1	
Barrière Sas entrée, haute	1	
Barrière Sas entrée, basse	1	
Barrière position basse au niveau de la voie PL n°1	1	

Dossier Spécification

Diagramme de contexte



Dictionnaire

Name: Gerer accès (systeme UC OS9)
Type: Context node

Name: Automate Principal
Type: Terminator

Name: Console maitre
Type: Terminator

Name: Disque dur Base donnees
Type: Terminator

Name: HTR
Type: Terminator

Name: Matrice Video
Type: Terminator

Name: Périphériques controle acces
Type: Terminator

Name: Reseau
Type: Terminator
Comment: Réseau Ethernet auquel sont connectés : système UC OS9, PC Unix SUPRA et PCs de Supervision

Name: arret
Type: Control flow

Name: etats_automate
Type: Discrete flow
Bnf: etats_capteurs + infos_modbus
Comment: informations, formattées MODBUS, réceptionnées par le système UC OS9 en provenance de l'automate principal (76 entrées TOR) cf Annexe B-1

Name: informations_mouvement
Type: Discrete flow
Bnf: [mouvement_realise | infos_mouvement]

Name: infos_acces
Type: Discrete flow
Bnf: [trame_NEDAP | trame_MTCA]
Comment: informations issues des boitiers NEDAP ou MTCA associés à des voies VL ou PL, en Entrée ou Sortie

Name: horloge
Type: Control flow
Comment: controle (frequence = 1Hz) activant une séquence d'interrogation de l'automate principal, la lecture de "etats_automate" et la sauvegarde de ces informations dans la zone tampon "etat_capteurs".

Name: marche
Type: Control flow

Name: mouvement_a_stocker
Type: Discrete flow
Bnf: [mouvement_prevu | mouvement_realise]

Name: ordres_automate
Type: Discrete flow
Bnf: ordres_pilot_autom + infos_modbus
Comment: ordres, formattés MODBUS, envoyés par le système UC OS9 vers l'automate principal (26 sorties TOR) cf Annexe B-1

Name: ordres_matrice
Type: Discrete flow
Bnf: ordres_video + en-tete
Comment: trame formattée contenant "ordres_video"

Name: trame_emise
Type: Discrete flow
Bnf: [trame_cmd_unix | trame_cmd_Os9 | trame_rep]
Comment: trames générées sur le réseau par le système UC OS9

Name: trame_recue
Type: Discrete flow
Bnf: [trame_cmd_unix | trame_cmd_Os9 | trame_rep]
Comment: trames recues via le réseau par le système UC OS9

DFD / DFC 0

trame_recue
dialoguer_ réseau 6
trame_emise
trame_unix_recue
trame_unix_emise
Dialoguer _unix 3
trame_supervision _reçue
trame_supervision _emise
gerer_bornes _accés 5
infos_acces
liste_mouvements _realises_jour
mouvement_prevu
id_acces
Dialoguer _supervision 1
informations_ mouvement
mouvement_realise
ordres_pilot _autom
gerer_base_ données 8
infos_mouvement
etats_recup
etats_capteurs
mouvement _a_stocker
ordres_video
etats_capteurs
horloge
Piloter _matrice 2
etat_modifie
Piloter _automate 4
ordres_ automate
etats_ automate
ordres_matrice
controler_0 7
arret
marche

Dictionnaire

Name: controler_0
Type: Control process

Name: dialoguer_reseau
Type: Process

Name: dialoguer_supervision
Type: Process

Name: dialoguer_unix
Type: Process

Name: gerer_base_donnees
Type: Process

Name: gerer_bornes_acces
Type: Process

Name: piloter_automate
Type: Process

Name: piloter_matrice
Type: Process

Name: etats_capteurs
Type: Discrete flow
Bnf: Select Auto Barriere VL_PL + Defaut Alim Barriere VL_PL_SAS + Infos Onduleurs + Infos Portail SNCF Nord + Infos Portail SNCF Centre + Infos Portail SNCF SUD ECM + Infos Portail SNCF Eng Parc + Infos Portail Reinjection + Infos Portail ES_Automate Terminal
Comment: 76 informations TOR issues de l'automate principal

Name: etats_capteurs
Type: Store
Bnf: etats_capteurs
Comment: zone tampon destinee au stockage des 76 entrees TOR rapatriees via l'automate principal

Name: etat_modifie
Type: Control flow
Comment: controle indiquant la fin de la reactualisation de la zone tampon "etats-capteurs" par le process "pilot_automate"

Name: etats_recup
Type: Discrete flow
Bnf: (Select Auto Barriere VL_PL) + (Defaut Alim Barriere VL_PL_SAS) + (Infos Onduleurs) + (Infos Portail SNCF Nord) + (Infos Portail SNCF Centre) + (Infos Portail SNCF SUD ECM) + (Infos Portail SNCF Eng Parc) + (Infos Portail Reinjection) + (Infos Portail ES_Automate Terminal)
Comment: informations lues dans la zone tampon

Name: id_acces
Type: Discrete flow
Bnf: type + numero_voie + [Numero_VL_PL_PAM | carte]
Comment: information identifiant en clair un véhicule PAM ou un badge ainsi que la voie VL_PL concernée et le sens du transfert

Name: infos_mouvement
Type: Discrete flow
Bnf: type + (data_heure) + [(carte) | (tracteur) | (nb_colis) + (0{colis}2)]
Comment: flot véhiculant certaines informations - extraites de la BD - spécifiques à un mouvement prévu par une société de manutention

Name: liste_mouvements_realises_jour
Type: Discrete flow
Bnf: 0 {mouvement_realise } 500
Comment: liste de mouvements effectivement réalisés, emise quotidiennement -à 0.00H - à destination du PC UNIX SUPRA

Name: mouvement_prevu
Type: Discrete flow
Bnf: mouvement
Comment: informations concernant un mouvement prévu par une société de manutention

Name: mouvement_realise
Type: Discrete flow
Bnf: mouvement
Comment: informations caractérisant un mouvement effectivement réalisé

Name: ordres_pilot_autom
Type: Discrete flow
Bnf: (Interphone VL_PL Pupitre) + (Ordre Ouverture Barriere VL_PL) + (Lance Magnetoscope) + (Ordre Engin Parc Cassien)
Comment: ordres émis par le système UC OS9 en direction de l'automate principal (28 sorties TOR) cf Annexe B-1

Name: ordres_video
Type: Discrete flow
Comment: informations permettant de piloter (sélection de canaux) la matrice vidéo (ex : association caméra X à moniteur Y ; patchwork: caméra X visualisée dans fenêtre N du moniteur Y)

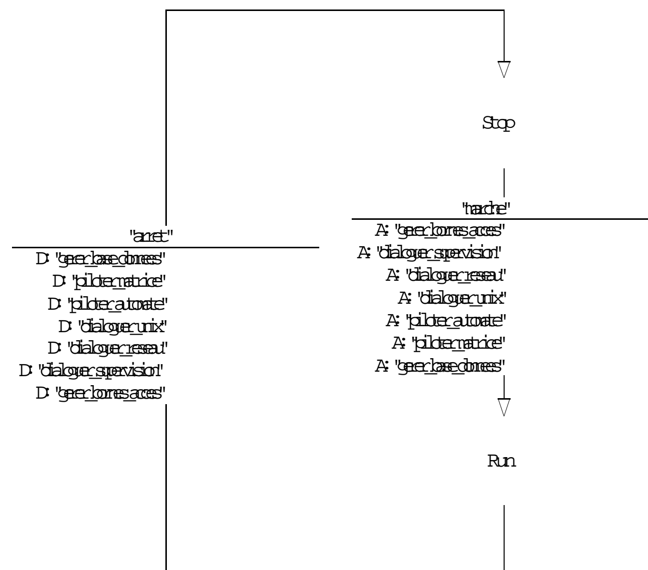
Name: trame_supervision_emise
Type: Discrete flow
Bnf: [trame_cmd_Os9 | trame_rep]
Comment: trame en emission support du dialogue système UC OS9 - PCs Supervision

Name: trame_supervision_recue
Type: Discrete flow
Bnf: [trame_cmd_Os9 | trame_rep]
Comment: trame en réception support du dialogue système UC OS9 - PCs Supervision

Name: trame_unix_emise
Type: Discrete flow
Bnf: [trame_cmd_unix | trame_rep]
Comment: trame en emission support du dialogue système UC OS9 - Unix SUPRA

Name: trame_unix_recue
Type: Discrete flow
Bnf: [trame_cmd_unix | trame_rep]
Comment: trame en reception support du dialogue Unix SUPRA - système UC OS9

CSPEC 0 : controler_0



Dictionnaire

Name: Stop
Type: State

Name: Run
Type: State

Dictionnaire suite – classement alphabétique-

Name: carte
Type: Discrete flow
Comment: numéro d'identification d'1 badge (12 caractères -> 4 : société, 7 : numéro carte, 1 : crc)

Name: colis
Type: Discrete flow
Comment: précise : nature (0: conteneur, 1: autre) + mode du transfert (0: normal, 1: transit, 2: douane) + identification conteneurs (13 caractères: 4-> société, 7 -> numéro conteneur, 1 -> crc) + nombre de conteneurs (appartient à [0,4]) + commentaires (2 x 80 caractères)

Name: crc
Type: Discrete flow

Name: data_heure
Type: Discrete flow
Comment: précise date et heure d'un mouvement

Name: Defaut Alim Barriere VL_PL_SAS
Type: Discrete flow
Comment: information binaire signalant un défaut d'alimentation barrière VL ou PL ou SAS

Name: en-tete
Type: Discrete flow
Comment: en-tête de trame

Name: entree_sortie
Type: Discrete flow
Comment: définit le sens du mouvement 0: entrée, 1: sortie

Name: err_type
Type: Discrete flow
Comment: contient le type de l'erreur identifiée par le process recepteur d'une requête

Name: infos_modbus
Type: Discrete flow
Comment: informations spécifiques au protocole modbus, encapsulant les données à transmettre

Name: Infos Onduleurs
Type: Discrete flow
Comment: informations TOR issue automate principal

Name: Infos Portail ES_Automate Terminal
Type: Discrete flow
Comment: informations TOR issue automate principal

Name: Infos Portail Reinjection
Type: Discrete flow
Comment: informations TOR issue automate principal

Name: Infos Portail SNCF Centre
Type: Discrete flow
Comment: informations TOR issue automate principal

Name: Infos Portail SNCF Eng Parc
Type: Discrete flow
Comment: informations TOR issue automate principal

Name: Infos Portail SNCF Nord
Type: Discrete flow
Comment: informations TOR issue automate principal

Name: Infos Portail SNCF SUD ECM
Type: Discrete flow
Comment: informations TOR issue automate principal

Name: infos_a_valider
Type: Discrete flow
Bnf: entree_sortie + tracteur + carte + nb_colis + (0{colis}2)
Comment: informations associées a un mouvement en entree ou sortie, qui devront etre controlees et validees par système UC OS9 afin d'autoriser le mouvement.

Name: infos_portillons
Type: Discrete flow
Comment: informations de : présence de vehicule sur voies PL + etat des barrieres ES_PL

Name: Interphone VL_PL Pupitre
Type: Discrete flow
Comment: ordre TOR destiné à automate principal (selection d'interphone)

Name: Lance Magnetoscope
Type: Discrete flow
Comment: ordre TOR destiné à automate principal

Name: login
Type: Discrete flow
Comment: nom user + password

Name: mouvement
Type: Discrete flow
Bnf: type + (data_heure) + (carte) + (tracteur) + (nb_colis) + (0{colis}2)
Comment: informations caractérisant un mouvement

Name: nb_colis
Type: Discrete flow
Comment: nombre de colis référencés dans un mouvement (appartient à [0, 2])

Name: Numero_VL_PL_PAM
Type: Discrete flow
Comment: numéro d'identification d'un véhicule PAM

Name: numero_voie
Type: Discrete flow
Comment: contient le numéro d'identification de la voie ou se présente le véhicule (VL ou PL ou VL_PL_PAM)

Name: Ordre Engin Parc Cassien
Type: Discrete flow
Comment: ordre TOR destiné à automate principal (gestion ouverture / fermeture du portail)

Name: Ordre Ouverture Barriere VL_PL
Type: Discrete flow
Comment: ordre TOR déstiné à automate principal (gestion ouverture / fermeture des barrières des voies VL, PL)

Name: prise_poste
Type: Discrete flow
Comment: information définissant la voie PL et les interphones affèctés au poste de supervision courant

Name: reponse
Type: Discrete flow
Comment: réponse à une requête : OK ou NON_OK

Name: Select Auto Barriere VL_PL
Type: Discrete flow
Comment: information TOR issue de l'automate principal (gestion sélection des barrières des voies VL, PL)

Name: tracteur
Type: Discrete flow
Comment: numéro d'immatriculation d'un PL

Name: trame_cmd_Os9
Type: Discrete flow
Bnf: type + [login | prise_poste | carte | infos_portillons | infos_a_valider]
Comment: trame permettant le dialogue UC OS9 - PC Supervision

Name: trame_cmd_unix
Type: Discrete flow
Bnf: type + login + (data_heure) + (carte) + (tracteur) + (nb_colis) + (0{colis}2) + crc
Comment: trame permettant le dialogue UC OS9 - Unix SUPRA

Name: trame_MTCA
Type: Discrete flow
Comment: trame issue boitier MTCA , associée à un badge + (une présence véhicule VL_PL sur une voie identifiée) + (état barrières VL_PL)

Name: trame_NEDAP
Type: Discrete flow
Comment: trame issue boitier NEDAP , associée à la présence d'un véhicule VL_PL_PAM sur une voie identifiée

Name: trame_rep
Type: Discrete flow
Bnf: reponse + err_type
Comment: trame de réponse à une requête vehiculée soit par une trame "trame_cmd_unix" soit par une "trame_cmd_Os9"

Name: type
Type: Discrete flow
Comment: type de la requête (entrée, sortie, test login, ...)

OS 9-68K

I) Table d'index :

I) Table d'index : 1
II) Présentation....1
Son architecture est structurée en couches : 1
III) Concepts principaux....2
1) Module mémoire : 2
2) Tâche....2
3 Communication entre tâches....2
Pipe : 2
Datamodule : 3
Signaux : 3
Evénement ("event") : 3

II) Présentation

OS968K est un système d'exploitation multitâches, multi-utilisateurs, il présente des caractéristiques de compacité et de facilité de mise en œuvre pour la génération des applications temps réel.

Ce S.E. est construit suivant la structure d'UNIX et lui est très comparable sur les points suivants :

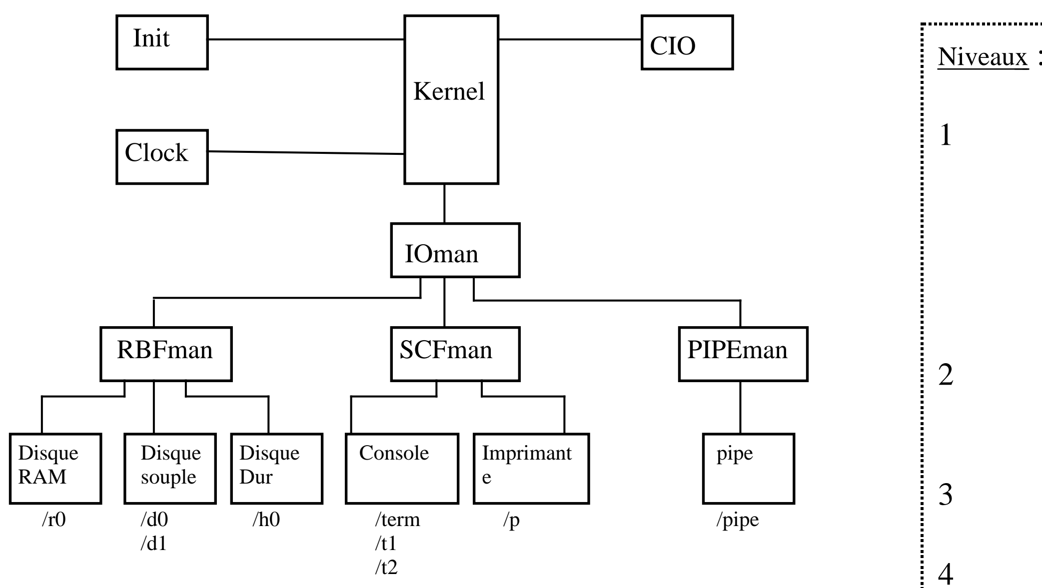
- Système unifié d'entrées/sorties
- Séquenceur de commande Shell (gestion pipe, redirection E/S, multiprogrammation, gestion environnement de programmation, …)
- Gestion du multitâche et des signaux
- Supporte l'environnement TCP/IP.

Son architecture est structurée en couches :

Le noyau exécutif (kernel) associé aux modules Init et Clock est responsable de l'exécution des fonctions système. C'est à dire la gestion du multitâche, des Entrées/Sorties au niveau 1, de l'accès aux autres modules systèmes.

Parmi les niveaux dédiés à la gestion des Entrées/Sorties , on distingue :

- niveau 2: les gestionnaires de fichiers (file manager) chargés de gérer un type de transfert(caractère, bloc, …).
- niveau 3: les gestionnaires de circuits (driver) chargés de piloter un circuit d'E/S particulier.
- niveau 4: les descripteurs de périphériques (file descriptor): petites tables de paramètres associées à un port et permettant de paramétrer son comportement.



III) Concepts principaux

1) Module mémoire :

Tous les fichiers exécutables et zones de mémoire partagée (datamodule) se présentent sous la forme de modules, décrits dans le fichier d'en-tête module.h et structurés comme suit :

En-tête
Corps (entry point) (contenant soit: - code exécutable - données)
CRC

2) Tâche

L'Ordonnanceur gère le temps CPU alloué à chaque tâche.
Les principaux états d'une tâche sont :

Prêt : La tâche est active et attend qu'un processeur soit libre pour s'exécuter.
En sommeil : La tâche est hors service pour une période de temps donné ou jusqu'à réception d'un signal. Des fonctions systèmes dédiées figent la tâche dans cet état (sleep(), …).

La création d'une tâche se fait par l'appel à la fonction système fork() qui crée un process autonome :
- disposant d'1 pile et d'1 zone de données
- héritant de certains canaux d'E/S de son père.

3 Communication entre tâches

Les stratégies les plus courantes utilisent :
- pipe
- zone de mémoire partagée (datamodule)

On pourrait aussi citer les signaux, normalement dédiés à la synchronisation mais susceptibles de véhiculer une faible quantité d'information (2 octets).

Pipe :

OS9 propose 2 types de pipes :

anonyme : - nécessite un lien de parenté entre tâches communicantes
- lors de sa fermeture, la mémoire est désallouée même si le pipe contient encore des informations - taille par défaut : 80 octets -

nommé : - ne nécessite pas de lien de parenté entre tâches communicantes
- lors de sa fermeture, la mémoire ne sera désallouée que lorsque le pipe ne contiendra plus d'information
- taille paramétrable à la création
- un processus qui demande une écriture dans un pipe nommé plein est mis en sommeil par OS9 (et sera réveillé dès libération de place).

Les entrées / sorties de type "pipe" sont gérées par trois modules particuliers :
- gestionnaire de fichiers : **pipeman**
  Il fait tout, il supporte la communication entre processus par pipe. Il est appelé à chaque requête d'entrée/sortie du descripteur /pipe
- driver : **null**
  Il ne fait rien
- descripteur: **pipe**
  A un pipe anonyme est associé /pipe, à un pipe nommé est associé /pipe/nom_pipe.

Datamodule :

Zone de mémoire partageable entre plusieurs tâches, l'accès au module doit être arbitré par l'application.

Signaux :

Fonctionnement analogue à Unix.

Evénement ("event") :

Concept permettant la synchronisation inter tâches.
Succinctement, ce concept repose sur :

- un état : ev_value : valeur courante de l'event
- des paramètres : wait_inc, signal_inc : valeurs ajoutées à ev_value lors respectivement, de la mise en attente d'un événement, de la signalisation d'un événement par un process.
  : ev_min,ev_max : valeurs définissant une fenêtre ; tant que ev_value est hors de cette fenêtre, le process ayant fait un appel à la fonction _ev_wait() reste en attente de cet événement.
- des fonctions:
  - _ev_creat() : création d'un évènement ("event")
  - _ev_signal() : signale un événement
  - _ev_wait() : mise en attente d'un événement
  - _ev_link() : création d'un lien sur un événement
  - _ev_unlink() : destruction d'un lien sur un événement

DOCUMENTATION SOCKET UNIX

1) Table d'index :

1) Table d'index : **1**
2) accept a connection on a socket : accept() **1**
3) bind a name to a socket : bind() **1**
4) initiate a connection on a socket : connect() **1**
5) get network host entry : gethostbyname, gethostbyaddr, gethostent, sethostent, endhostent, herror **2**
6) get socket name : getsockname **3**
7) get and set options on sockets : getsockopt, setsockopt **3**
8) listen for connections on a socket : listen **3**
9) receive a message from a socket : recv, recvfrom **4**
10) send a message from a socket : send, sendto **4**
11) shut down part of a full-duplex connection: shutdown **5**
12) create an endpoint for communication : socket **5**
13) Divers. **6**

2) accept a connection on a socket : accept()

int accept(int s, struct sockaddr *addr, int *addrlen);

The argument **s** is a socket that has been created with socket(2), bound to an address with bind(2), and is listening for connections after a listen(2). The accept argument extracts the first connection request on the queue of pending connections, creates a new socket with the same properties of s and allocates a new file descriptor for the socket. If no pending connections are present on the queue, and the socket is not marked as non blocking, accept blocks the caller until a connection is present. If the socket is marked non-blocking and no pending connections are present on the queue, accept returns an error as described below. The accepted socket may not be used to accept more connections. The original socket s remains open.

The argument **addr** is a result parameter that is filled in with the address of the connecting entity, as known to the communications layer. The exact format of the addr parameter is determined by the domain in which the communication is occurring.

The **addrlen** is a value-result parameter; it should initially contain the amount of space pointed to by addr; on return it will contain the actual length (in bytes) of the address returned. This call is used with connection-based socket types, currently with SOCK_STREAM.

RETURN VALUES

The call returns -1 on error. If it succeeds, it returns a non-negative integer that is a descriptor for the accepted socket.

3) bind a name to a socket : bind()

int bind(int sockfd, struct sockaddr *my_addr, int addrlen);

DESCRIPTION

bind gives the socket, **sockfd**, the local address **my_addr**.

my_addr is **addrlen** bytes long. Traditionally, this is called "assigning a name to a socket" (when a socket is created with socket(2), it exists in a name space (address family) but has no name assigned.)

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and errno is set appropriately.

4) initiate a connection on a socket : connect()

int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);

DESCRIPTION

The parameter **sockfd** is a socket. If it is of type SOCK_DGRAM, this call specifies the peer with which the socket is to be associated; this address is that to which datagrams are to be sent, and the only address from which datagrams are to be received. If the socket is of type SOCK_STREAM, this call attempts to make a connection to another socket. The other socket is specified by **serv_addr**, which is an address in the communications space of the socket. Each communications space interprets the **serv_addr**, parameter in its own way. Generally, stream sockets may successfully connect only once; datagram sockets may use connect multiple times to change their association. Datagram sockets may dissolve the association by connecting to an invalid address, such as a null address.

RETURN VALUE

If the connection or binding succeeds, zero is returned. On error, -1 is returned, and errno is set appropriately.

5) get network host entry : gethostbyname, gethostbyaddr, gethostent, sethostent, endhostent, herror

```
#include <netdb.h>
extern int h_errno;
struct hostent *gethostbyname(char *name)
struct hostent *gethostbyaddr(char *addr, int len, int type)
struct hostent *gethostent()
sethostent(int stayopen)
endhostent()
herror(char *string)
```

DESCRIPTION

Gethostbyname and **gethostbyaddr** each return a pointer to an object with the following structure describing an internet host referenced by name or by address, respectively. This structure contains either the information obtained from the name server, named(8), or broken-out fields from a line in /etc/hosts. If the local name server is not running these routines do a lookup in /etc/hosts.

```
Struct hostent {
        char    *h_name;        /* official name of host */
        char    **h_aliases;    /* alias list */
        int     h_addrtype;     /* host address type */
        int     h_length;       /* length of address */
        char    **h_addr_list;  /* list of addresses from name server */
        };
#define         h_addr  h_addr_list[0]  /* address, for backward compatibility */
```

The members of this structure are:

h_name : Official name of the host.
h_aliases : A zero terminated array of alternate names for the host.
h_addrtype : The type of address being returned; currently always AF_INET.
h_length : The length, in bytes, of the address.
h_addr_list : A zero terminated array of network addresses for the host. Host addresses are returned in network byte order.
h_addr : The first address in h_addr_list; this is for backward compatiblity.

When using the nameserver, **gethostbyname** will search for the named host in the current domain and its parents unless the name ends in a dot. If the name contains no dot, and if the environment variable ``HOSTALIASES'' contains the name of an alias file, the alias file will first be searched for an alias matching the input name. See hostname(7) for the domain search procedure and the alias file format.

Sethostent may be used to request the use of a connected TCP socket for queries. If the stayopen flag is non-zero, this sets the option to send all queries to the name server using TCP and to retain the connection after each call to gethostbyname or gethostbyaddr. Otherwise, queries are performed using UDP datagrams.

Endhostent closes the TCP connection.

DIAGNOSTICS

Error return status from gethostbyname and gethostbyaddr is indicated by return of a null pointer. The external integer h_errno may then be checked to see whether this is a temporary failure or an invalid or unknown host. The routine herror can be used to print an error message describing the failure. If its argument string is non NULL, it is printed, followed by a colon and a space. The error message is printed with a trailing newline.

h_errno can have the following values:

HOST_NOT_FOUND No such host is known.
TRY_AGAIN This is usually a temporary error and means that the local server did not receive a response from an authoritative server. A retry at some later time may succeed.
NO_RECOVERY Some unexpected server failure was encountered. This is a non recoverable error.
NO_DATA The requested name is valid but does not have an IP address; this is not a temporary error. This means that the name is known to the name server but there is no address associated with this name. Another type of request to the name server using this domain name will result in an answer; for example, a mailforwarder may be registered for this domain.

FILES

/etc/hosts

6) get socket name : getsockname

int getsockname(int s , struct sockaddr * name , int * namelen)

DESCRIPTION

Getsockname returns the current name for the specified socket. The **namelen** parameter should be initialized to indicate the amount of space pointed to by **name**. On return it contains the actual size of the name returned (in bytes).

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and errno is set appropriately. 0 is returned if the call succeeds, -1 if it fails.

7) get and set options on sockets : getsockopt, setsockopt

int getsockopt(int s, int level, int optname, void *optval, int *optlen);
int setsockopt(int s, int level, int optname, const void *optval, int optlen);

DESCRIPTION

Getsockopt and **setsockopt** manipulate the options associated with a socket. Options may exist at multiple protocol levels; they are always present at the uppermost socket level.

When manipulating socket options the level at which the option resides and the name of the option must be specified. To manipulate options at the socket level, level is specified as SOL_SOCKET. To manipulate options at any other level the protocol number of the appropriate protocol controlling the option is supplied. For example, to indicate that an option is to be interpreted by the TCP protocol, level should be set to the protocol number of TCP; see getprotoent(3).

The parameters optval and optlen are used to access option values for setsockopt. For getsockopt they identify a buffer in which the value for the requested option(s) are to be returned. For getsockopt, optlen is a value-result parameter, initially containing the size of the buffer pointed to by optval, and modified on return to indicate the actual size of the value returned. If no option value is to be supplied or returned, optval may be NULL.

Optname and any specified options are passed uninterpreted to the appropriate protocol module for interpretation.

The include file sys/socket.h contains definitions for socket level options, described below. Options at other protocol levels vary in format and name; consult the appropriate entries in section 4 of the manual.

Most socket-level options utilize an int parameter for optval. For setsockopt, the parameter should be non-zero to enable a boolean option, or zero if the option is to be disabled. SO_LINGER uses a struct linger parameter, defined in sys/socket.h, which specifies the desired state of the option and the linger interval (see below).

SO_SNDTIMEO and SO_RCVTIMEO use a struct timeval parameter, defined in sys/time.h.

The following options are recognized at the socket level.

Except as noted, each may be examined with getsockopt and set with setsockopt .

SO_DEBUG	enables recording of debugging information
SO_REUSEADDR	enables local address reuse
SO_KEEPALIVE	enables keep connections alive
SO_DONTROUTE	enables routing bypass for outgoing messages
SO_LINGER	linger on close if data present
SO_BROADCAST	enables permission to transmit broadcast messages
SO_OOBINLINE	enables reception of out-of-band data in band
SO_SNDBUF	set buffer size for output
SO_RCVBUF	set buffer size for input
SO_SNDLOWAT	set minimum count for output
SO_RCVLOWAT	set minimum count for input
SO_SNDTIMEO	set timeout value for output
SO_RCVTIMEO	set timeout value for input
SO_TYPE	get the type of the socket (get only)
SO_ERROR	get and clear error on the socket (get only)

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and errno is set appropriately.

8) listen for connections on a socket : listen

int listen(int s, int backlog);

DESCRIPTION

To accept connections, a socket is first created with socket(2), a willingness to accept incoming connections and a queue limit for incoming connections are specified with **listen**, and then the connections are accepted with accept(2). The listen call applies only to sockets of type SOCK_STREAM or SOCK_SEQPACKET.

The **backlog** parameter defines the maximum length the queue of pending connections may grow to. If a connection request arrives with the queue full the client may receive an error with an indication of ECONNREFUSED, or, if the underlying protocol supports retransmission, the request may be ignored so that retries may succeed.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and errno is set appropriately.

9) receive a message from a socket : recv, recvfrom

int recv(int s, void *buf, int len, unsigned int flags);
int recvfrom(int s, void *buf, int len, unsigned int flags, struct sockaddr *from, int *fromlen);

DESCRIPTION

Recvfrom is used to receive messages from a socket, and may be used to receive data on a socket whether or not it is connection-oriented.

If **from** is non-nil, and the socket is not connection oriented, the source address of the message is filled in.

Fromlen is a value-result parameter, initialized to the size of the buffer associated with from, and modified on return to indicate the actual size of the address stored there.

The **recv** call is normally used only on a connected socket (see connect(2)) and is identical to recvfrom with a nil from parameter. As it is redundant, it may not be supported in future releases.

All three routines return the length of the message on successful completion. If a message is too long to fit in the supplied buffer, excess bytes may be discarded depending on the type of socket the message is received from (see socket(2)).

If no messages are available at the socket, the receive call waits for a message to arrive, unless the socket is nonblocking (see fcntl(2)) in which case the value -1 is returned and the external variable errno set to EWOULDBLOCK. The receive calls normally return any data available, up to the requested amount, rather than waiting for receipt of the full amount requested; this behavior is affected by the socket-level options SO_RCVLOWAT and SO_RCVTIMEO described in getsockopt(2).

The select(2) call may be used to determine when more data arrive.

The flags argument to a recv call is formed by or'ing one or more of the values:

MSG_OOB	process out-of-band data
MSG_PEEK	peek at incoming message
MSG_WAITALL	wait for full request or error

RETURN VALUES

These calls return the number of bytes received, or -1 if an error occurred.

10) send a message from a socket : send, sendto

int send(int s, const void *msg, int len , unsigned int flags);
int sendto(int s, const void *msg, int len unsigned int flags, const struct sockaddr *to, int tolen);

DESCRIPTION

Send, **sendto** are used to transmit a message to another socket. Send may be used only when the socket is in a connected state, while sendto may be used at any time.

The address of the target is given by **to** with **tolen** specifying its size. The length of the message is given by **len**. If the message is too long to pass atomically through the underlying protocol, the error EMSGSIZE is returned, and the message is not transmitted.

No indication of failure to deliver is implicit in a send.

Locally detected errors are indicated by a return value of -1.

If no messages space is available at the socket to hold the message to be transmitted, then send normally blocks, unless the socket has been placed in non-blocking I/O mode. The select(2) call may be used to determine when it is possible to send more data.

The **flags** parameter may include one or more of the following:

```
#define    MSG_OOB          0x1    /* process out-of-band data */
#define    MSG_DONTROUTE    0x4    /* bypass routing, use direct interface */
```

RETURN VALUES

The call returns the number of characters sent, or -1 if an error occurred.

11) shut down part of a full-duplex connection: shutdown

int shutdown(int s, int how);

DESCRIPTION

The **shutdown** call causes all or part of a full-duplex connection on the socket associated with **s** to be shut down.
If **how** is 0, further receives will be disallowed. If **how** is 1, further sends will be disallowed. If **how** is 2, further sends and receives will be disallowed.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and errno is set appropriately.

12) create an endpoint for communication : socket

int socket(int domain, int type, int protocol);

DESCRIPTION

Socket creates an endpoint for communication and returns a descriptor.

The **domain** parameter specifies a communications domain within which communication will take place; this selects the protocol family which should be used. These families are defined in the include file sys/socket.h. The currently understood formats are :

AF_UNIX	(UNIX internal protocols)
AF_INET	(ARPA Internet protocols)
AF_ISO	(ISO protocols)
AF_NS	(Xerox Network Systems protocols)
AF_IMPLINK	(IMP "host at IMP" link layer)

The socket has the indicated **type**, which specifies the semantics of communication. Currently defined types are:

SOCK_STREAM
SOCK_DGRAM
SOCK_RAW
SOCK_SEQPACKET
SOCK_RDM

A SOCK_STREAM type provides sequenced, reliable, two-way connection based byte streams. An out-of-band data transmission mechanism may be supported. A SOCK_DGRAM socket supports datagrams (connectionless, unreliable messages of a fixed (typically small) maximum length). A SOCK_SEQPACKET socket may provide a sequenced, reliable, two-way connection-based data transmission path for datagrams of fixed maximum length; a consumer may be required to read an entire packet with each read system call. This facility is protocol specific, and presently implemented only for PF_NS. SOCK_RAW sockets provide access to internal network protocols and interfaces. The types SOCK_RAW, which is available only to the super-user, and SOCK_RDM, which is planned, but not yet implemented, are not described here.
The protocol specifies a particular protocol to be used with the socket. Normally only a single protocol exists to support a particular socket type within a given protocol family. However, it is possible that many protocols may exist, in which case a particular protocol must be specified in this manner. The protocol number to use is particular to the "communication domain" in which communication is to take place; see protocols(5).
Sockets of type SOCK_STREAM are full-duplex byte streams, similar to pipes. A stream socket must be in a connected state before any data may be sent or received on it. A connection to another socket is created with a connect(2) call. Once connected, data may be transferred using read(2) and write(2) calls or some variant of the send(2) and recv(2) calls. When a session has been completed a close(2) may be performed. Out-of-band data may also be transmitted as described in send(2) and received as described in recv(2).
The communications protocols used to implement a SOCK_STREAM insure that data is not lost or duplicated.
If a piece of data for which the peer protocol has buffer space cannot be successfully transmitted within a reasonable length of time, then the connection is considered broken and calls will indicate an error with -1 returns and with ETIMEDOUT as the specific code in the global variable errno. The protocols optionally keep sockets warm by forcing transmissions roughly every minute in the absence of other activity. An error is then indicated if no response can be elicited on an otherwise idle connection for a extended period (e.g. 5 minutes). A SIGPIPE signal is raised if a process sends on a broken stream; this causes naive processes, which do not handle the signal, to exit.
SOCK_SEQPACKET sockets employ the same system calls as SOCK_STREAM sockets. The only difference is that read(2) calls will return only the amount of data requested, and any remaining in the arriving packet will be discarded.
SOCK_DGRAM and SOCK_RAW sockets allow sending of datagrams to correspondents named in send(2) calls. Datagrams are generally received with recvfrom(2), which returns the next datagram with its return address.

An fcntl(2) call can be used to specify a process group to receive a SIGURG signal when the out-of-band data arrives.
It may also enable non-blocking I/O and asynchronous notification of I/O events via SIGIO.
The operation of sockets is controlled by socket level options. These options are defined in the file sys/socket.h. Setsockopt(2) and getsockopt(2) are used to set and get options, respectively.

RETURN VALUES
A -1 is returned if an error occurs, otherwise the returnvalue is a descriptor referencing the socket.

13) Divers.

FICHIERS D'ENTETE

```
#include <sys/types.h>
#include <sys/socket.h>
```

STRUCTURES

```
Struct in_addr { u_long s_addr;};
struct   sockaddr_in {
        short       sin_family;         /* famille de l'adresse : AF_INET, .. */
        u_short     sin_port;           /* numéro port */
        struct      in_addr  sin_addr;  /* adresse internet */
        char        sin_zero[8];        /* champ de 8 octets nuls */
};
```

FONCTIONS

u_long	htonl(u_long)	passage de la représentation locale d'un entier long en sa représentation réseau
u_short	htons(u_short)	passage de la représentation locale d'un entier court en sa représentation réseau
u_long	ntohl(u_long)	passage de la représentation réseau d'un entier long en sa représentation locale
u_short	ntohs(u_short)	passage de la représentation réseau d'un entier court en sa représentation locale

DOCUMENTATION INETD

NAME

inetd - internet ``super-server''

SYNOPSIS

`inetd [-d] [-R rate] [configuration file]`

DESCRIPTION

The inetd program should be run at boot time by /etc/rc (see rc(8)). It then listens for connections on certain internet sockets. When a connection is found on one of its sockets, it decides what service the socket corresponds to, and invokes a program to service the request. The server program is invoked with the service socket as its standard input, output and error descriptors. After the program is finished, inetd continues to listen on the socket (except in some cases which will be described below). Essentially, inetd allows running one daemon to invoke several others, reducing load on the system.

The options available for inetd:

- -d Turns on debugging.
- -R rate Specifies the maximum number of times a service can be invoked in one minute; the default is 1000.

Upon execution, inetd reads its configuration information from a configuration file which, by default, is **/etc/inetd.conf**. There must be an entry for each field of the configuration file, with entries for each field separated by a tab or a space. Comments are denoted by a ``#'' at the beginning of a line. There must be an entry for each field. The fields of the configuration file are as follows:

service name
socket type
protocol
wait/nowait
user
server program
server program arguments

There are two types of services that inetd can start: standard and TCP_MUX. A standard service has a well-known port assigned to it; it may be a service that implements an official Internet standard or is a BSD-specific service. As described in RFC 1078, TCPMUX services are nonstandard services that do not have a well-known port assigned to them. They are invoked from inetd when a program connects to the ``tcpmux'' wellknown port and specifies the service name. This feature is useful for adding locally-developed servers.

The *service-name* entry is the name of a valid service in the file /etc/services. For ``internal'' services (discussed below), the service name must be the official name of the service (that is, the first entry in /etc/services). For TCPMUX services, the value of the service-name field consists of the string ``tcpmux'' followed by a slash and the locally-chosen service name. The service names listed in */etc/services* and the name ``help'' are reserved. Try to choose unique names for your TCP_MUX services by prefixing them with your organization's name and suffixing them with a version number.

The *socket-type* should be one of ``stream'', ``dgram'', ``raw'', ``rdm'', or ``seqpacket'', depending on whether the socket is a stream, datagram, raw, reliably delivered message, or sequenced packet socket. TCPMUX services must use ``stream''.

The *protocol* must be a valid protocol as given in /etc/protocols. Examples might be ``tcp'' or ``udp''. TCPMUX services must use ``tcp''.

The *wait/nowait* entry specifies whether the server that is invoked by inetd will take over the socket associated with the service access point, and thus whether inetd should wait for the server to exit before listening for new service requests. Datagram servers must use ``wait'', as they are always invoked with the original datagram socket bound to the specified service address. These servers must read at least one datagram from the socket before exiting. If a datagram server connects to its peer, freeing the socket so inetd can received further messages on the socket, it is said to be a ``multi-threaded'' server; it should read one datagram from the socket and create a new socket connected to the peer. It should fork, and the parent should then exit to allow inetd to check for new service requests to spawn new servers. Datagram servers which process all incoming datagrams on a socket and eventually time out are said to be ``single-threaded''. Comsat(8), (biff(1)) and talkd(8) are both examples of the latter type of datagram server. Tftpd(8) is an example of a multi-threaded datagram server.

Servers using stream sockets generally are multi-threaded and use the ``nowait" entry. Connection requests for these services are accepted by id, and the server is given only the newly-accepted socket connected to a client of the service. Most stream-based services operate in this manner. Stream-based servers that use ``wait" are started with the listening service socket, and must accept at least one connection request before exiting. Such a server would normally accept and process incoming connection requests until a timeout. TCPMUX services must use ``nowait".

The *user* entry should contain the user name of the user as whom the server should run. This allows for servers to be given less permission than root.

The *server-program* entry should contain the pathname of the program which is to be executed by inetd when a request is found on its socket. If inetd provides this service internally, this entry should be ``internal".

The *server_program_arguments* should be just as arguments normally are, starting with argv[0], which is the name of the program. If the service is provided internally, the word ``internal" should take the place of this entry.

The inetd program provides several ``trivial" services internally by use of routines within itself. These services are ``echo", ``discard", ``chargen" (character generator), ``daytime" (human readable time), and ``time" (machine readable time, in the form of the number of seconds since midnight, January 1, 1900). All of these services are tcp based.
For details of these services, consult the appropriate RFC from the Network Information Center.

The inetd program rereads its configuration file when it receives a hangup signal, SIGHUP. Services may be added, deleted or modified when the configuration file is reread.

RFC 1078 describes the TCPMUX protocol: ``A TCP client connects to a foreign host on TCP port 1. It sends the service name followed by a carriage-return line-feed <CRLF>. The service name is never case sensitive.
The server replies with a single character indicating positive (+) or negative (-) acknowledgment, immediately followed by an optional message of explanation, terminated with a <CRLF>. If the reply was positive, the selected protocol begins; otherwise the connection is closed." The program is passed the TCP connection as file descriptors 0 and 1.

If the TCPMUX service name begins with a ``+", id returns the positive reply for the program. This allows you to invoke programs that use stdin/stdout without putting any special server code in them.

The special service name ``help" causes inetd to list TCPMUX services in inetd.conf.

EXAMPLES

Here are several example service entries for the various types of services:

```
ftp                 stream  tcp  nowait  root   /usr/libexec/ftpd          ftpd -l
ntalk               dgram   udp  wait    root   /usr/libexec/ntalkd        ntalkd
tcpmux/+date        stream  tcp  nowait  guest  /bin/date                  date
tcpmux/phonebook    stream  tcp  nowait  guest  /usr/local/bin/phonebook   phonebook
```

ERROR MESSAGES

The inetd server logs error messages using syslog(3). Important error messages and their explanations are:

- service/protocol server failing (looping), service terminated.

The number of requests for the specified service in the past minute exceeded the limit. The limit exists to prevent a broken program or a malicious user from swamping the system. This message may occur for several reasons:

1) there are lots of hosts requesting the service within a short time period
2) a 'broken' client program is requesting the service too frequently
3) a malicious user is running a program to invoke the service in a 'denial of service' attack
4) the invoked service program has an error that causes clients to retry quickly.

Use the [-R] option, as described above, to change the rate limit. Once the limit is reached, the service will be reenabled automatically in 10 minutes.

- service/protocol: No such user 'user', service ignored
- service/protocol: getpwnam : user : No such user

No entry for user exists in passwd file. The first messages occurs when inetd (re)reads the configuration file. The second message occurs when the service is invoked.

- service: can't set uid number
- service: can't set gid number

The user or group ID for the entry's user is invalid.

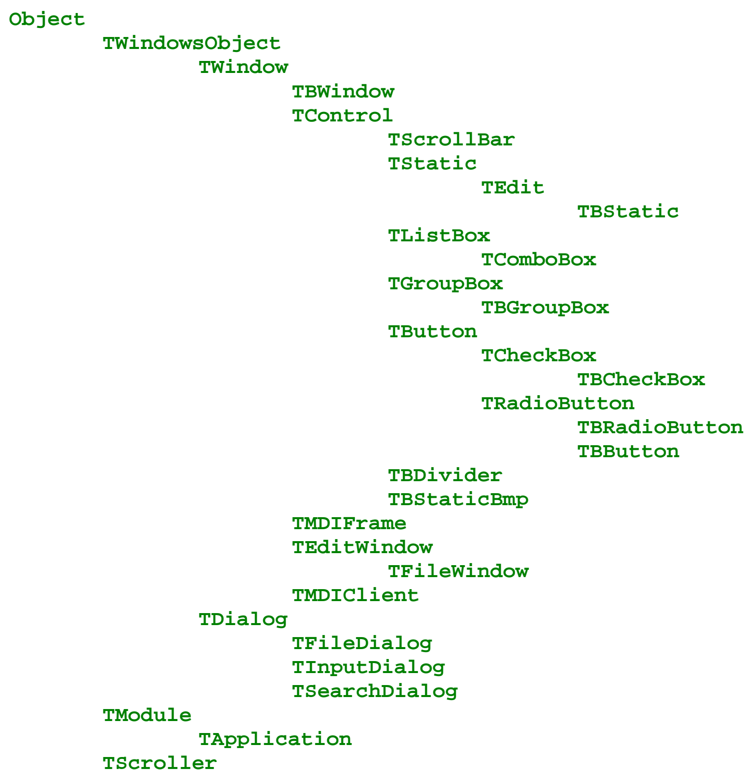
Documentation Borland 3.1 pour Windows (OWL 1.0)

I) Table d'index :

I) Table d'index : 1
II) Hiérarchie des Classes ObjectWindows 1
1) Object (classe abstraite) (OBJECT.H) 1
2) Présentation de la classe TWindowsObject (WINDOBJ.H) 2
3) Présentation de la classe TWindow (WINDOW.H) 2
4) Présentation de la classe TControl (CONTROL.H) 2
5) Présentation des classes TButton (BUTTON.H) 3
6) Présentation de la classe TDialog (DIALOG.H) 3
8) Présentation de la classe TApplication (APPLICAT.H) 4
III) Messages Windows 5
1) Les Messages (liste partielle) (3.1) 5
2) La demande immédiate d'un message SendMessage (2.x) 5
3) Le postage d'un message en différé PostMessage (2.x) 5

II) Hiérarchie des Classes ObjectWindows

Dans ce diagramme hiérarchique, les liens de parenté vont de gauche à droite:
Object est parent de TWindowsObject, de TModule et de TScroller
TWindowsObject est parent de TDialog et de TWindow
TWindow est parent de TBWindow, de TControl, de TMDIFrame, de TEditWindow et de TMDIClient,



```
Object
        TWindowsObject
                TWindow
                        TBWindow
                        TControl
                                TScrollBar
                                TStatic
                                        TEdit
                                                TBStatic
                                TListBox
                                        TComboBox
                                TGroupBox
                                        TBGroupBox
                                TButton
                                        TCheckBox
                                                TBCheckBox
                                        TRadioButton
                                                TBRadioButton
                                                TBButton
                                TBDivider
                                TBStaticBmp
                        TMDIFrame
                        TEditWindow
                                TFileWindow
                        TMDIClient
                TDialog
                        TFileDialog
                        TInputDialog
                        TSearchDialog
        TModule
                TApplication
        TScroller
```

TWindowsObject et TScroller sont hérités de TStreamable.

1) Object (classe abstraite) (OBJECT.H)

La classe Object est une classe abstraite qui constitue la base de toute la hiérarchie des classes conteneurs orientée objets (à l'exception des classes itérateurs).

Ses méthodes fournissent les fonctions élémentaires pour toutes les classes dérivées et les objets qu'elles contiennent.

2) Présentation de la classe TWindowsObject (WINDOBJ.H)

TWindowsObject et une classe abstraite, dérivée de Object, définissant le comportement de base que partagent tous les objets d'interface d'ObjectWindows (fenêtres, boîtes de dialogue et contrôles).

TWindowsObject gère une liste des fenêtres filles dans sa liste des enfants et induit un comportement qui s'applique à cette liste des enfants. TWindowsObject définit les fonctions membres donnant l'accès aux fenêtres présentes dans sa liste des enfants.

TWindowsObject joue également un rôle important dans la mise en correspondance des messages entrants et des fonctions membres qui leur répondent.

3) Présentation de la classe TWindow (WINDOW.H)

La classe TWindow est dérivée de TWindowsObject et définit les comportements spécifiques aux fenêtres. (TDialog s'intéresse aux comportements spécifiques aux boîte de dialogues). Les fonctions TWindow permettent de définir les paramètres propres à la création d'une fenêtre, y compris le recensement et les attributs.

TWindow est une fenêtre générique complète contenant un titre dont la position et la taille peuvent être modifiées. Vous pouvez construire et créer des instances de TWindow mais le plus souvent vous utiliserez TWindow comme base pour vos classes de fenêtre spécialisées.

Vous pouvez utiliser une instance ou une classe dérivée de TWindow comme une fenêtre enfant MDI; il vous suffit pour cela de spécifier une fenêtre TMDIFrame comme parent.

De nombreuses fonctions membres TWindow fournissent les descripteurs nécessaires à l'utilisation d'un objet TScroller, ce qui facilite le défilement de l'objet TWindow.

TWindow est également la classe de base de TMDIClient,, un contrôle spécialisé dans les applications compatibles MDI.

Constructeurs et Destructeur

TWindow(PTWindowsObject AParent, LPSTR ATitle, PTModule AModule = NULL);
TWindow(HWND AnHWindow);
TWindow(StreamableInit);
virtual ~TWindow( );

Champs

TWindow::Attr
TWindow::FocusChildHandle
TWindow::Scroller

Méthodes

TWindow::ActivationResponse
TWindow::AssignMenu
TWindow::build
TWindow::Create
TWindow::GetClassName
TWindow::GetWindowClass
TWindow::isA
TWindow::nameOf
TWindow::Paint
TWindow::read
TWindow::SetupWindow
TWindow::~TWindow
TWindow::TWindow
TWindow::WMCreate
TWindow::WMHScroll
TWindow::WMLButtonDown
TWindow::WMMDIActivate
TWindow::WMMove
TWindow::WMPaint
TWindow::WMSize
TWindow::WMVScroll
TWindow::write

4) Présentation de la classe TControl (CONTROL.H)

TControl permet d'unifier les classes contrôle dérivées telles que TScrollBar et TButton.
Les objets contrôles des classes dérivées permettent de représenter des composants de contrôle de l'interface Windows.
Un objet contrôle:

- doit être utilisé pour créer dans un parent TWindow.
- permet de faciliter la communication entre l'application et les composants de contrôles d'un objet TDialog.

Constructeurs

TControl(PTWindowsObject AParent, int AnId, LPSTR ATitle, int X, int Y, int W, int H,
PTModule ATModule = NULL);
TControl(PTWindowsObject AParent, int ResourceId, PTModule ATModule = NULL);
TControl(StreamableInit);

Méthodes
TControl::GetId
TControl::ODADrawEntire
TControl::ODAFocus
TControl::ODASelect
TControl::TControl
TControl::WMDrawItem
TControl::WMPaint

5) Présentation des classes TButton (BUTTON.H)

TButton est un objet interface incarnant un bouton action indépendant dans Windows. TButton permet de créer un contrôle bouton dans un TWindow. Il sert aussi à faciliter les communications entre application et contrôles d'un objet TDialog.

Il existe deux modèles de boutons:

- un bouton standard avec bordure fine et
- un bouton préactivé avec bordure épaisse (frappe de Enter).

Il ne peut bien sûr y avoir qu'un seul bouton préactivé par fenêtre.

Constructeurs
TButton(PTWindowsObject AParent, int AnId, LPSTR AText, int X,int Y,int W,int H, BOOL IsDefault, PTModule AModule=NULL);
TButton(PTWindowsObject AParent, int ResourceId, PTModule AModule = NULL);
TButton(StreamableInit);

Champs
TButton::IsDefPB

Méthodes
TButton::BMSetStyle
TButton::build
TButton::GetClassName
TButton::SetupWindow
TButton::TButton
TButton::WMGetDlgCode

6) Présentation de la classe TDialog (DIALOG.H)

Les objets TDialog incarnent des éléments interface du type boîte de dialogue aussi bien modale que non modale. Un objet TDialog possède une définition de ressource correspondante qui décrit la position et l'aspect de chacun des contrôles qu'elle possède. L'identificateur de la définition de ressource est fournie au constructeur de l'objet TDialog.

Un objet TDialog est associé avec un élément d'interface modal ou non modal par un appel à l'une de ces méthodes Execute ou Create respectivement. Ces méthodes ne sont pas normalement appelées directement dans une application ObjectWindows. Les méthodes ExecDialog et MakeWindow de TModule sont appelées pour tester la validité de l'objet TDialog avant son association.

Une boîte modale interdit toute action dans la fenêtre parente pendant qu'elle est ouverte.

Constructeurs et Destructeur
TDialog(PTWindowsObject AParent, LPSTR AName, PTModule AModule = NULL);
TDialog(PTWindowsObject AParent, int ResourceId, PTModule AModule = NULL);
TDialog(StreamableInit);
virtual~TDialog();

Champs
TDialog::Attr
TDialog::IsModal

Méthodes
TDialog::build
TDialog::Cancel
TDialog::CloseWindow
TDialog::Create
TDialog::Destroy
TDialog::Execute
TDialog::GetClassName
TDialog::GetItemHandle
TDialog::GetWindowClass
TDialog::isA
TDialog::nameOf
TDialog::Ok
TDialog::read
TDialog::SendDlgItemMsg
TDialog::SetCaption
TDialog::SetupWindow

TDialog::ShutDownWindow
TDialog::~TDialog
TDialog::TDialog
TDialog::WMClose
TDialog::WMInitDialog
TDialog::WMQueryEndSession
TDialog::write

7) Présentation de la classe TModule (MODULE.H)

Les librairies DLL (Dynamic Link Libraries) d'ObjectWindows construisent une instance de TModule, laquelle agit comme un objet de remplacement pour le module librairie.

Les application ObjectWindows construisent une instance de TApplication,, dérivée de TModule.

TModule définit le comportement commun à la librairie et aux modules d'application. Les fonctrions membres de TModule permettent de gérer la mémoire nécessaire aux fenêtres ainsi que les erreurs.

Constructeur et Destructeur

TModule(LPSTR AName, HANDLE LibhInstance, LPSTR LiblpCmdLine);
virtual ~TModule();

Champs

TModule::hInstance
TModule::IpCmdLine
TModule::Name
TModule::Status

Méthodes

TModule::Error
TModule::ExecDialog
TModule::GetClientHandle
TModule::GetParentObject
TModule::hashValue
TModule::isA
TModule::isEqual
TModule::LowMemory
TModule::MakeWindow
TModule::nameOf
TModule::printOn
TModule::RestoreMemory
TModule::~TModule
TModule::TModule
TModule::ValidWindow

8) Présentation de la classe TApplication (APPLICAT.H)

TApplication est dérivée de TModule et sert de remplacement orienté-objets à un module applicatif Windows classique.

TApplication et TModule fournissent les capacités de base d'une application Windows. Les méthodes de TApplication assurent l'initialisation des instances et le traitement des messages.

Constructeur et Destructeur

TApplication(LPSTR AName, HANDLE AnInstance, HANDLE APrevInstance, LPSTR ACmdLine, int ACmdShow);
~TApplication();

Champs

TApplication::HAccTable
TApplication::hPrevInstance
TApplication::KBHandlerWnd
TApplication::MainWindow
TApplication::nCmdShow

Méthodes

TApplication::CanClose
TApplication::IdleAction
TApplication::InitApplication
TApplication::InitInstance
TApplication::InitMainWindow
TApplication::isA
TApplication::MessageLoop
TApplication::nameOf
TApplication::ProcessAccels
TApplication::ProcessAppMsg
TApplication::ProcessDlgMsg

TApplication::ProcessMDIAccels
TApplication::Run
TApplication::SetKBHandler
TApplication::~TApplication
TApplication::TApplication

III) Messages Windows

1) Les Messages (liste partielle) (3.1)

BM_GETCHECK	Retrieves the check state of a button
BM_GETSTATE	Retrieves the state of a button
BM_SETCHECK	Sets the check state of a button
BM_SETSTATE	Sets the highlight state of a button
BM_SETSTYLE	Changes the style of a button
WM_COMMAND	Specifies a command message
WM_ENABLE	Indicates when enable state of window is changing
WM_QUIT	Requests that an application be terminated
WM_SYSCOMMAND	Indicates when a System-command is requested
WM_USER	Indicates a range of message values

2) La demande immédiate d'un message SendMessage (2.x)

```
LRESULT SendMessage(hwnd, uMsg, wParam, lParam)
    HWND      hwnd;      // handle of destination window
    UINT      uMsg;      // message to send
    WPARAM    wParam;    // first message parameter
    LPARAM    lParam;    // second message parameter
```

The SendMessage function sends the specified message to the given window or windows. The function calls the window procedure for the window and does not return until that window procedure has processed the message. This is in contrast to the PostMessage function, which places (posts) the message in the window's message queue and returns immediately.

Parameter	Description
hwnd	Identifies the window to which the message will be sent. If this parameter is HWND_BROADCAST, the message will be sent to all top-level windows, including disabled or invisible unowned windows.
uMsg	Specifies the message to be sent.
wParam	Specifies 16 bits of additional message-dependent information.
lParam	Specifies 32 bits of additional message-dependent information.

Returns

The return value specifies the result of the message processing and depends on the message sent.

Comments

If the message is being sent to another application and the wParam or lParam parameter is used to pass a handle or pointer to global memory, the memory should be allocated by the GlobalAlloc function using the GMEM_SHARE flag.

Example

The following example calls the SendMessage function to send an EM_SETSEL message to a multiline edit control, telling it to select all the text. It then calls SendMessage to send a WM_COPY message to copy the selected text to the clipboard.

```
SendMessage(hwndMle, EM_SETSEL, 0, MAKELONG(0, -1));
SendMessage(hwndMle, WM_COPY, 0, 0L);
```

3) Le postage d'un message en différé PostMessage (2.x)

```
BOOL PostMessage(hwnd, uMsg, wParam, lParam)
    HWND      hwnd;      // handle of the destination window
    UINT      uMsg;      // message to post
    WPARAM    wParam;    // first message parameter
    LPARAM    lParam;    // second message parameter
```

The PostMessage function posts (places) a message in a window's message queue and then returns without waiting for the corresponding window to process the message. Messages in a message queue are retrieved by calls to the GetMessage or PeekMessage function.

Parameter	Description
hwnd	Identifies the window to which the message will be posted. If this parameter is HWND_BROADCAST, the message will be posted to all top-level windows, including disabled or invisible unowned windows.
uMsg	Specifies the message to be posted.

wParam	Specifies 16 bits of additional message-dependent information.
lParam	Specifies 32 bits of additional message-dependent information.

Returns

The return value is nonzero if the function is successful. Otherwise, it is zero.

Comments

An application should never use the PostMessage function to post a message to a control.

If the message is being posted to another application, and the wParam or lParam parameters are used to pass a handle or pointer to a global memory object, the memory should be allocated by the GlobalAlloc function, using the GMEM_SHARE flag.

The PostMessage function fails if the message queue for the receiving application is full. This is especially likely if an application posts several messages without allowing the receiving task to run. The GetMessage, PeekMessage, and WaitMessage functions yield control to other applications.

ANNEXE PROGRAMMES WINDOWS

I) Table d'index :

I) Table d'index : 1
II) Fichiers communs. 1
1) Fichier types.h 1
2) Fichier util.h 3
3) Fichier util.cpp 3
4) Fichier de ressource bitmap.rc 4
5) Fichier hosts 5
III) Fichier demandes manutentionnaires (commanditaires) 5
1) Fichier manut.cpp (uniquement les déclarations) 5
2) Fichier de ressources associé 7
IV) Fichier supervision (voies PL, ...) 7
1) Fichier superv_pl.cpp 7
2) Fichier de ressources associé 17

Remarques:

Les fichiers fournis sont partiels.
Tous les traitements d'erreurs ne sont pas donnés.
Seul la gestion des voies PL est traitée.

II) Fichiers communs.

1) Fichier types.h

```
//****************************************************
const unsigned long color16[16] = {
      0x000000lu, 0x00007flu, 0x007f00lu, 0x007f7flu,
      0x7f0000lu, 0x7f007flu, 0x7f7f00lu, 0x7f7f7flu,
      0xc0c0c0lu, 0x0000fflu, 0x00ff00lu, 0x00ffffflu,
      0xff0000lu, 0xff00fflu, 0xffff00lu, 0xfffffflu};
//****************************************************

#define PORT_CMD_OS9   5000
#define PORT_REP_OS9   5001
#define PORT_UNIX      5002

#define LOGIN_NAME     12
#define LOGIN_PASSWD    6

typedef struct Login
  {
   char nom   [LOGIN_NAME];
   char passwd[LOGIN_PASSWD];
  } LOGIN;

typedef struct Prise_Poste
  {
   unsigned int mot1, // contient le numéro de la voie sélectionnée
                mot2; // contient la liste des interphones sélectionnés
  } PRISE_POSTE;

typedef struct Infos_Portillons
  {
   union
    {
     unsigned int    data;   // pour atteindre le mot entier
     struct
      {
   int b_entree       : 1, // bit d'état barrière  1: ouvert, 0: fermé
       b_entree_PL1   : 1,
       b_entree_PL2   : 1,
       b_entree_VL    : 1,
       b_sortie_PL1   : 1
```

```
       b_sortie_PL2   : 1,
       b_sortie_VL    : 1,
       b_sortie       : 1,
       p_entree       : 1, // bit d'état capteur présence  1: véhicule présent, 0: rien
       p_entree_PL1   : 1,
       p_entree_PL2   : 1,
       p_entree_VL    : 1,
       p_sortie_PL1   : 1,
       p_sortie_PL2   : 1,
       p_sortie_VL    : 1,
       p_sortie       : 1;
      } bit;
     };

  } INFOS_PORTILLONS;

char *ERR_INFOS[]={"OK","Erreur nom login","Erreur password","Erreur carte non valide",
   "Erreur tracteur_interdit","Erreur liste colis1","Erreur liste colis2",
   "Erreur date non valide","erreur indéfinie"};

#define OK                             0x0000
#define LOGIN_OK                       0x0000
#define ERR_NOM_INCONNU                0x0001
#define ERR_PASSWD                     0x0002
#define ERR_CARTE_NON_VALIDE           0x0004
#define ERR_TRACTEUR_INTERDIT          0x0008
#define ERR_LISTE_COlIS_1              0x0010
#define ERR_LISTE_COlIS_2              0x0020
#define ERR_DATE_NON_VALIDE            0x0040
#define ERR_INDEFINIE                  0x0080
#define ERR_CRC                        0x0100
#define ERR_INDEFINIE                  0x8000


#define TYPE_LOGIN                            1
#define TYPE_PRISE_POSTE                      2
#define TYPE_ARRIVE_TRACTEUR                  3
#define TYPE_DEMANDE_VALIDATION               4
#define TYPE_DEMANDE_INFOS_PORTILLONS         5
#define TYPE_REP_INFOS_PORTILLONS             6
#define TYPE_DEMANDE_OUVERTURE                7
#define TYPE_DEMANDE_REINJECTION              8
#define TYPE_CONVOI_ENTREE                    20
#define TYPE_CONVOI_SORTIE                    21

#define REP_OK                 0
#define REP_ERR                1

#define LCARTE                 13
#define LTRACTEUR              10
#define MAX_CONTENEURS         4
#define LSTRING_CONTENEUR      14 // AAAA 00000000 0 - NOM:4 NUMERO:8 CRC:1 ==> implique de supprimer les blancs
#define LCOMMENTAIRE           80

typedef struct Colis
  {
   unsigned char      nature;        // 0: contenaire       1: autre
   unsigned char      mode;          // 0: normal           1: transit     2: douane
   unsigned char      nb_conteneurs; // 0..4
   char               liste[MAX_CONTENEURS][LSTRING_CONTENEUR];    // liste des conteneurs
   char               cmt [LCOMMENTAIRE],    // commentaires
                      cmtm[LCOMMENTAIRE];    // commentaires manutentionnaire
  } COLIS;

typedef struct Colis_Mnt
  {
   unsigned char      nature;        // 0: conteneur        1: autre
   unsigned char      mode;          // 0: normal           1: transit     2: douane
   unsigned char      nb_conteneurs; // 0..4
   char               liste[MAX_CONTENEURS][LSTRING_CONTENEUR];    // liste des conteneur
   char               cmtm[LCOMMENTAIRE];    // commentaires manutentionnaire
  } COLIS_MNT;

typedef struct infos_a_valider
  {
   unsigned char      entree_sortie;        // 0: entree          1: sortie
```

```
   char               tracteur[LTRACTEUR];  // 9999 AAA 13  ou 9999AAA13 ==> implique de supprimer les
blancs
   char               carte   [LCARTE];            // arrivee d'un badge, envoyé par le serveur
   unsigned char      nb_colis;
  } INFOS_A_VALIDER;

typedef struct Message_A_Valider
  {
   INFOS_A_VALIDER    *infos_a_valider;
   COLIS              *colis1,
                      *colis2;
  } MESSAGE_A_VALIDER;

typedef struct Trame_Cmd_OS9
  {
   int                type;                 // nature des messages TYPE_????
   union
     {
      LOGIN           login;                // contient le  login du client
      PRISE_POSTE     prise_poste;          // contient les infos de prise de poste du client
      char            carte[LCARTE];        // arrivee d'un badge, envoyé par le serveur
      INFOS_PORTILLONSinfos_portillons;     // modifications ou demande d'infos,  envoyé par le serveur
      INFOS_A_VALIDER infos_a_valider;      // demande de validation par le client
     };
} TRAME_CMD_OS9;

typedef struct Trame_rep
  {
   unsigned int       reponse;
   unsigned int       err_type;
} TRAME_REP;

  typedef struct Date_Heure
  {
   int          heure  :6;
   int          jour   :6;
   int          mois   :4;
   int          annee  :16;
  } DATE_HEURE;

 typedef struct Trame_Cmd_Unix
  {
   unsigned char      type;
   LOGIN              login;                // contient le  login du client
   DATE_HEURE         date_heure;           // jj mm aa
   char               carte[LCARTE];        // arrivee d'un badge, envoyé par le serveur
   char               tracteur[LTRACTEUR];  // 9999 AAA 13  ou 9999AAA13
   unsigned char      nb_colis;
   COLIS_MNT          colis1,
                      colis2;
   unsigned int       crc;                  // contrôle de trame
  } TRAME_CMD_UNIX;
```

2) Fichier util.h

```
int bissextile(int annee);
int numero_jour(int annee, int mois, int jour);
void modif_jour(int annee, int mois, int *jour, int val);
void jour_numero(int annee, int n_jours, int *mois, int *jour);
unsigned int calculer_crc(unsigned char *trame, int size);
char *sup_blancs(char *texte);
```

3) Fichier util.cpp

```
#include "util.h"

static int jour_mois [2] [13] = {
     {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
     {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}};

//****************************************************
int bissextile(int annee)
{
// ??????????????????????
}

//****************************************************
```

```
void modif_jour(int annee, int mois, int *jour, int val)
 /* ajoute(+/-) val à jour et contrôle la validité (avec retour à 1 ou au max auto)*/
{
 int bis, j=*jour+val;

 bis = bissextile(annee);
 if(j > jour_mois[bis][mois]) j-=jour_mois[bis][mois];
 else if(j < 1)               j+=jour_mois[bis][mois];
 if((j >= 1) && (j <= jour_mois[bis][mois])) *jour = j;
}

//***************************************************
int numero_jour(int annee, int mois, int jour)
 /* donne le nombre de jours écoulés depuis le début de l'année*/
{
 int i, bis;

 bis = bissextile(annee);
 for(i = 1; i < mois; i++) jour += jour_mois [bis] [i];
 return(jour);
}

//***************************************************
void jour_numero(int annee, int n_jours, int *mois, int *jour)
 /* donne le jour et le mois de l'année écoulés depuis n_jours*/
{
 int i, bis;

 bis = bissextile(annee);
 for(i = 1; n_jours > jour_mois [bis] [i]; i++)
    n_jours -= jour_mois [bis] [i];
 if(n_jours <= 31)
   {
    *mois = i;
    *jour = n_jours;
   }
}

//***************************************************
unsigned int calculer_crc(unsigned char *trame, int size)
{
// ??????????????????
}

//***************************************************
char *sup_blancs(char *texte)
{
 char *s=texte, *d=texte;
 for(; (*d = *s) != 0; s++) if(*s != ' ') d++;
 return(texte);
}
```

4) Fichier de ressource bitmap.rc

```
BITMAP_TRACTEUR BITMAP
BEGIN
// ????????????
END

BITMAP_1COLIS BITMAP
BEGIN
// ????????????
END

BITMAP_2COLIS BITMAP
BEGIN
// ????????????
END

BITMAP_BARRIERE_LEVEE BITMAP
BEGIN
// ????????????
END


BITMAP_BARRIERE_BAISSEE BITMAP
BEGIN
```

```
// ????????????
END
```

5) Fichier hosts (Dans le fichier \windows\hosts pour PC_OP1)

```
127.0.0.1           localhost
128.1.10.126        UC_MAR
128.1.4.83          UC_GARE
128.1.10.121        SUPRAMAR
128.1.2.1           SUPRAFOS
128.6.1.101         UC_MOUREPIANE
128.6.1.102         UD_MOUREPIANE
128.6.1.103         PC_OP1              LOCAL
128.6.1.104         PC_OP2
128.6.1.105         PC_OP3
128.6.1.106         PC_CARTE
128.6.1.107         SUPRA_MOUREPIANE
```

III) Fichier demandes manutentionnaires (commanditaires).

1) Fichier manut.cpp (uniquement les déclarations).

```
#define WIN31
#define _CLASSDLL
#include <owl.h>
#include <window.h>
#include <control.h>
#include <winsock.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <dos.h>
#pragma hdrstop

#include "types.h"
#include "util.h"

/************************************************************************/
/*                   DECLARATION DU TYPE de Com_Trame_Unix              */
class Com_Trame_Unix
{ // cette classe gère la communication PC_Manutentionnaire et UNIX_SUPRA
 protected:
  HWND           fenetre;
  WSADATA        wasadata;
  int            sock_init;
  SOCKET         sock_client;
  unsigned char  ip_local [ 4], ip_serv [ 4];
  char           str_local[16], str_serv[16];
  TRAME_CMD_UNIX trame_cmd;
  TRAME_REP      trame_rep;
  struct sockaddr_in addr_serveur_cmd;
  int                addr_taille;

 public:
                         Com_Trame_Unix(HWND afenetre);
                         ~Com_Trame_Unix();
 virtual unsigned long   valider(unsigned char type);
 virtual unsigned long   test_passwd()      {return(valider(TYPE_LOGIN));};
 virtual char *          addr_str_serveur() {return(st_serv );};
 virtual char *          addr_str_client () {return(st_local);};
 virtual void            nom_passwd(char *nom, char *passwd);
 virtual int             tracteur(char *texte);
 virtual int             carte   (char *texte);
 virtual int             date    (int heure, int jour, int mois, int annee);
 virtual int             nb_colis(int val);
 virtual int             nature(int n_colis, int val);
 virtual int             mode  (int n_colis, int val);
 virtual int             nb_conteneurs(int n_colis, int val);
 virtual int             liste        (int n_colis, int num, char *texte);
 virtual int             commentaires (int n_colis, char *texte);
};

/************************************************************************/
```

```
/*                    DECLARATION DU TYPE de TCamion                         */
class TCamion : public TControl
{// cette classe gère l'affichage du tracteur et des colis
 protected:
     int x1, y1, x2, y2, *transport;

 public:
                TCamion(PTWindowsObject AParent, int AnId, int ax1, int ay1,
                int ax2, int ay2, int *atransport);
 virtual void   DefWndProc  (TMessage & msg);
 virtual void   WMPaint(RTMessage Msg) = [WM_FIRST + WM_PAINT];
 virtual void   affiche ();
};

/************************************************************************/
/*                    DECLARATION DU TYPE de TDialog_E_S_PL_Com             */
class TDialog_E_S_PL_Com  : public TDialog
{// cette classe gère la boite de dialogue des demandes d'E/S des commenditaires
protected:
  TCamion *tcamion;
  unsigned char choix;
  int     transport;
  int      combo_220, combo_320;
  int      heure, jour, mois, annee;
  Com_Trame_Unix *com_trame_unix;

 public:
               TDialog_E_S_PL_Com(PTWindowsObject aparent,
                              unsigned char achoix, Com_Trame_Unix *acom_trame_unix);
 virtual void  DefWndProc      (TMessage & msg);
 virtual void  WMInitDialog    (TMessage &msg) = [WM_FIRST + WM_INITDIALOG];
 virtual void  valider         (TMessage &msg) = [ID_FIRST + 130];
 virtual void  suivant         (TMessage &msg) = [ID_FIRST + 131];
 virtual int   cb_DM_GETDEFID  (int id_cb);
 virtual void  enable_convoi   (int val);
 virtual void  enable_colis    (int id_cb, int val);
 virtual int   remplir_colis   (int n_colis, int id_cb);
 virtual void  remplir_date_heure();
 virtual int   controler_date_heure();
};

/************************************************************************/
/*                    DECLARATION DU TYPE de TDialog_Login          */
class TDialog_Login : public TDialog
{// cette classe gère la boite de dialogue de la saisie du mot de passe pour la session
 protected:
   Com_Trame_Unix *com_trame_unix;

 public:
              TDialog_Login(PTWindowsObject aparent,Com_Trame_Unix *acom_trame_unix);
 virtual void  WMInitDialog    (TMessage &msg) = [WM_FIRST+WM_INITDIALOG];
 virtual void  Ok              (TMessage &msg) = [ID_FIRST + IDOK];
};

/************************************************************************/
/*                    DECLARATION DU TYPE de TFenetrePrincipale     */
#define cm_Quitter    400
#define cm_login      100
#define cm_sortie     102
#define cm_entree     101
#define cm_Aide       300

class TFenetrePrincipale : public TWindow
{
 protected: /* variables */
  int              var_connexion;
  Com_Trame_Unix   *com_trame_unix;

 public: /* Methodes */
                TFenetrePrincipale(PTWindowsObject aparent, LPSTR atitle,
                           LPSTR amenu, PTModule pmModule);
 virtual void   GetWindowClass (WNDCLASS _FAR & awndclass);
 virtual void   DefWndProc  (TMessage & msg);
 virtual void   SetupWindow ();
 virtual BOOL   CanClose    ();
 virtual void   WMClose     (RTMessage Msg) = [WM_FIRST + WM_CLOSE];
 virtual void   quitter     (TMessage & ) = [CM_FIRST + cm_Quitter ];
```

```
 virtual void   Login       (TMessage & ) = [CM_FIRST + cm_login];
 virtual void   Entree      (TMessage & ) = [CM_FIRST + cm_entree];
 virtual void   Sortie      (TMessage & ) = [CM_FIRST + cm_sortie];
 virtual void   aide        (TMessage & ) = [CM_FIRST + cm_Aide    ];

 virtual void   detruire_objets();
};

//***********************************************************************
//                   DECLARATION DU TYPE de Tmyapplication
class tmyapplication : public TApplication
{
 protected: // variables
   LPSTR nom_fenetre;    // nom de la fenêtre

 public: // Methodes
              tmyapplication(LPSTR lpszName, HANDLE ahInstance, HANDLE ahPrevInstance,
                     LPSTR lpszCmdLine, intnCmdShow);
 virtual void  InitMainWindow();
};
```

2) Fichier de ressources associé

```
#include "bitmap.rc"

MENU_CLIENT MENU
BEGIN
        MENUITEM "&Login", 100
        MENUITEM "&Entrée_Contenaires", 101, GRAYED
        MENUITEM "&Sortie_Contenaires", 102, GRAYED
        MENUITEM "&Quitter", 400
        MENUITEM "&Aide", 300
END

ICONE_CLIENT ICON
BEGIN
// ???????????????????
END

DIALOG_LOGIN DIALOG 18, 18, 212, 92
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "MOUREPIANE [LOGIN]"
BEGIN
        EDITTEXT 101, 53, 21, 76, 12, ES_LEFT | WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP
        CONTROL "", 100, "EDIT", ES_LEFT | ES_READONLY | WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP, 53, 4, 77, 12
        LTEXT "Nom:", -1, 34, 24, 19, 8, WS_CHILD | WS_VISIBLE | WS_GROUP
        LTEXT "Addr. IP poste:", -1, 4, 6, 49, 8, WS_CHILD | WS_VISIBLE | WS_GROUP
        CONTROL "", 102, "EDIT", ES_LEFT | ES_PASSWORD | WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP, 53, 40, 23, 12
        LTEXT "Mot de Passe:", -1, 6, 42, 47, 9, WS_CHILD | WS_VISIBLE | WS_GROUP
        DEFPUSHBUTTON "&Validation", 1, 16, 67, 38, 14, WS_CHILD | WS_VISIBLE | WS_TABSTOP
        PUSHBUTTON "&QUITTER", 2, 79, 67, 38, 14, WS_CHILD | WS_VISIBLE | WS_TABSTOP
END

DIALOG_E_S_PL DIALOG 19, 20, 295, 211
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Dialogue Entrée Sortie Commanditaire"
BEGIN
« « « « « « « « « « « « « « « « «
END
```

IV) Fichier supervision (voies PL, ...)

1) Fichier superv_pl.cpp (partiel)

```
#define WIN31
#define _CLASSDLL
#include <owl.h>
#include <window.h>
#include <control.h>
#include <winsock.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#pragma hdrstop

HANDLE  HInstance; // mém. de l'instance en global permet d'accéder au menus, icones, ...
#include "types.h"
#include "util.h"
```

```
/************************************************************************/
/*                   DECLARATION DU TYPE de TCamion                */
class TCamion : public TControl
{// cette classe gère l'affichage du tracteur et des colis
 protected:
     int x1, y1, x2, y2, *transport, *portillons;
     INFOS_PORTILLONS *infos_portillons;

 public:
              TCamion(PTWindowsObject AParent, int AnId, int ax1, int ay1,
                int ax2, int ay2, int *atransport,
                int *aportillons, INFOS_PORTILLONS *ainfos_portillons);
 virtual void   DefWndProc  (TMessage & msg);
 virtual void   WMPaint(RTMessage Msg) = [WM_FIRST + WM_PAINT];

 virtual void   affiche ();
};

//**********************************************************************
//                   DEFINITION DES METHODES de TCamion
// ???????????????????

/************************************************************************/
/*                   DECLARATION DU TYPE de TDialog_Sortie_PL            */
class TDialog_Sortie_PL  : public TDialog
{// cette classe gère la boite de dialogue des contrôles d'E/S (sur voie PL)
 protected:
  TCamion *tcamion;
  int     mot1, transport, portillons[2], capt_presence;
  int     combo_220, combo_320;
  char    carte[LCARTE];
  INFOS_PORTILLONS infos_portillons;

 public:
             TDialog_Sortie_PL(PTWindowsObject aparent, int amot1);
 virtual void  DefWndProc      (TMessage & msg);
 virtual void  WMInitDialog    (TMessage &msg) = [WM_FIRST + WM_INITDIALOG];
 virtual void  valider         (TMessage &msg) = [ID_FIRST + 130];
 virtual void  ouvrir          (TMessage &msg) = [ID_FIRST + 132];
 virtual void  reinjection     (TMessage &msg) = [ID_FIRST + 131];
 virtual int   cb_DM_GETDEFID  (int id_cb);
 virtual void  enable_convoi   (int val);
 virtual void  enable_colis    (int id_cb, int val);
 virtual void  remplir_colis   (COLIS *colis, int id_cb);
};

//**********************************************************************
//                   DEFINITION DES METHODES de  TDialog_Sortie_PL
TDialog_Sortie_PL::TDialog_Sortie_PL(PTWindowsObject aparent,int amot1)
        : TDialog(aparent,((amot1==3)||(amot1==4))?"DIALOG_SORTIE_PL":"DIALOG_ENTREE_PL",NULL)
{
 combo_220, combo_320 = 0;
 tcamion    = NULL;
 transport = 0;
 switch(mot1=amot1)
   {// Initialisation de portillon, de capt_presence et du titre
    // ???????????????????
   }
 infos_portillons.data = 0;
}

//**************************************************
void TDialog_Sortie_PL::DefWndProc  (TMessage & msg)
{
 switch(msg.Message)
   {
    // ??????????????????? : gestion des combox (id : 220, 320)et des radios boutons (id :101-103,221,321)
    case WM_USER+1 :
      switch(msg.WParam)
        {
         case TYPE_ARRIVE_TRACTEUR :
           if(((infos_portillons.data >> capt_presence) & 1) == 1)  // vérifie présence véhicule !!!
             {
              if(!transport) enable_convoi(1);
              SetDlgItemText(HWindow,122,(char *) msg.LParam);
              }
```

```
           else MessageBox(HWindow,"Erreur: pas de présence de véhicupe\nAppelez la maintenance",
             "Arrivée d'un badge",MB_OK);
          break;
        case TYPE_REP_INFOS_PORTILLONS :
          {
           unsigned int old;

           old = infos_portillons.data;
           infos_portillons.data = (int) msg.LParam;

           // contrôle que les infos modifiées concernent la presence tracteur
           if((old & (1 << capt_presence)) != (infos_portillons.data & (1 << capt_presence)))
             {
              if(((infos_portillons.data >> capt_presence)&1) == 1)
                {
                     if(!transport) enable_convoi(1);
                }
               else if( transport) enable_convoi(0);
              }
           // contrôle que les infos modifiées concernent la voie
           if(tcamion &&       ((old & ((1 << portillons[0])|(1 << portillons[1]))) !=
              (infos_portillons.data & ((1 << portillons[0])|(1 << portillons[1])))))
             { // affiche infos portillons modifiées
              InvalidateRect(tcamion->HWindow,NULL,TRUE); // toute la fenêtre doit être repeinte
              UpdateWindow  (tcamion->HWindow);           // envoit d'un message PAINT
             }
          }
          break;
        default : break;
        }
      break;
    default :
      TDialog::DefWndProc(msg);
      break;
   }
}

//***************************************************
void  TDialog_Sortie_PL::enable_convoi   (int val)
{// gestion ( validation non validation des contrôles) de la fenêtre de dialogue en fonction :
 // de la détection d'un tracteur (badgeage carte)
 // et des radios boutons (id :101-103) (tracteur seul, 1 colis, 2 colis)
// ??????????????????
}

//***************************************************
void  TDialog_Sortie_PL::enable_colis    (int id_cb, int val)
{ // complète enable_convois pour les contrôles des colis
// ??????????????????
}

//***************************************************
int TDialog_Sortie_PL::cb_DM_GETDEFID  (int id_cb)
{// gère la validation de la saisie des numéros de conteneurs dans les combox (320 et 321)
// ??????????????????
}

//***************************************************
// procédure lancée automatiquement après la construction de la fenêtre
void TDialog_Sortie_PL::WMInitDialog(TMessage & msg)
{
 TDialog::WMInitDialog(msg); // lancement de la procédure standard
 GetApplication()->MakeWindow(tcamion=new TCamion(this,104, 44*2, 8*2, 244*2, 40*2,&transport,
     portillons, &infos_portillons));
 SendDlgItemMsg(201,BM_SETCHECK,TRUE,0l);
 SendDlgItemMsg(211,BM_SETCHECK,TRUE,0l);
 SendDlgItemMsg(221,BM_SETCHECK,TRUE,0l);
 SendDlgItemMsg(301,BM_SETCHECK,TRUE,0l);
 SendDlgItemMsg(311,BM_SETCHECK,TRUE,0l);
 SendDlgItemMsg(321,BM_SETCHECK,TRUE,0l);
 PostMessage(Parent->HWindow,WM_USER+1,TYPE_DEMANDE_INFOS_PORTILLONS,0l);
}

//***************************************************
void TDialog_Sortie_PL::remplir_colis   (COLIS *colis, int id_cb)
{ // remplit la structure colis (1 ou 2) en fonctions des saisies opérateurs
 // ??????????????????
```

```
}

//**************************************************
void TDialog_Sortie_PL::valider(TMessage & msg)
{ // demande de validation des saisies opérateurs et du badgeage
 // ??????????????????
   {
    INFOS_A_VALIDER infos_a_valider;
    COLIS           colis1, colis2;
    MESSAGE_A_VALIDER message_a_valider={&infos_a_valider,&colis1,&colis2};
    char            texte[80];
    unsigned long   rep;

infos_a_valider.entree_sortie = (mot1 == 3) || (mot1 == 4);
    GetDlgItemText(HWindow,121,texte,80); // saisie de la plaque du tracteur
    strcpy(infos_a_valider.tracteur,sup_blancs(texte));
    if((SendDlgItemMsg(103,BM_GETCHECK,0,0l) & 1) == 1)
      { // 2 colis
       infos_a_valider.nb_colis = 2;
       remplir_colis(&colis1,200);
       remplir_colis(&colis2,300);
      }
    if((SendDlgItemMsg(102,BM_GETCHECK,0,0l) & 1) == 1)
      { // 1 colis
       infos_a_valider.nb_colis = 1;
       remplir_colis(&colis1,200);
      }
     else infos_a_valider.nb_colis = 0;

    if((rep=SendMessage(Parent->HWindow,WM_USER+1,TYPE_DEMANDE_VALIDATION,
               (long) &message_a_valider)) == 0l)
      { // valider ouverture
       EnableWindow(GetItemHandle(131),FALSE);
       EnableWindow(GetItemHandle(132),TRUE);
      }
     else
      { // valider ré injection, manque de préciser la nature de l'erreur
       sprintf(texte,"%lx",rep);
       MessageBox(HWindow,"","REPONSE VALIDATION: Ouverture non autorisées",MB_OK);
       EnableWindow(GetItemHandle(132),FALSE);
      }
   }
 msg.Result=0;
}

//**************************************************
void TDialog_Sortie_PL::ouvrir(TMessage & msg)
{
 SendMessage(Parent->HWindow,WM_USER+1,TYPE_DEMANDE_OUVERTURE,0l);
 msg.Result=0;
}

//**************************************************
void TDialog_Sortie_PL::reinjection(TMessage & msg)
{
 SendMessage(Parent->HWindow,WM_USER+1,TYPE_DEMANDE_REINJECTION,0l);
 msg.Result=0;
}

/***********************************************************************/
/*                  DECLARATION DU TYPE de TDialog_Login          */
class TDialog_Login : public TDialog
{// cette classe gère la boite de dialogue de la saisie du mot de passe pour la session
 protected:
  LOGIN         *plogin;
  unsigned char *ip_serv, *ip_local;

 public:
              TDialog_Login(PTWindowsObject aparent,unsigned char *aip_local,
                unsigned char *aip_serv, LOGIN *aplogin);
 virtual void  WMInitDialog    (TMessage &msg) = [WM_FIRST+WM_INITDIALOG];
 virtual void  Ok              (TMessage &msg) = [ID_FIRST + IDOK];
};

//**********************************************************************
//                  DEFINITION DES METHODES de  TDialog_Login
// ??????????????????
```

```
/***********************************************************************/
/*                  DECLARATION DU TYPE de TDialog_PrisePoste          */
class TDialog_PrisePoste  : public TDialog
{// cette classe gère la boite de dialogue de la saisie du choix de la prise de poste
 protected:
  PRISE_POSTE  *prise_poste;

 public:
              TDialog_PrisePoste(PTWindowsObject aparent,PRISE_POSTE  *aprise_poste);
 virtual void  WMInitDialog    (TMessage &msg) = [WM_FIRST+WM_INITDIALOG];
 virtual void  Ok              (TMessage &msg) = [ID_FIRST + IDOK];
};

//**********************************************************************
//                  DEFINITION DES METHODES de  TDialog_PrisePoste
// ??????????????????

/***********************************************************************/
/*                  DECLARATION DU TYPE de TFenetrePrincipale           */
#define cm_Quitter           100
#define cm_connexion         101
#define cm_deconnexion       102
#define cm_priseposte        103
#define cm_Aide              300
#define BLKSIZE              128

class TFenetrePrincipale : public TWindow
{
 protected: /* variables */
  int           var_connexion;
  PRISE_POSTE   prise_poste;
  WSADATA       wasadata;
  int           sock_init;
  SOCKET        sock_client_cmd, sock_client_rep;
  unsigned char ip_local[4], ip_serv[4];
  TDialog_Sortie_PL *dialog_sortie_pl;

 public: /* Methodes */
              TFenetrePrincipale(PTWindowsObject aparent, LPSTR atitle,
                          LPSTR amenu, PTModule pmModule);
 virtual void  GetWindowClass (WNDCLASS _FAR & awndclass);
 virtual void  DefWndProc  (TMessage & msg);
 virtual void  SetupWindow ();
 virtual void  WMClose     (RTMessage Msg) = [WM_FIRST + WM_CLOSE];
 virtual void  quitter     (TMessage & ) = [CM_FIRST + cm_Quitter ];
 virtual void  connexion   (TMessage & ) = [CM_FIRST + cm_connexion];
 virtual void  deconnexion (TMessage & ) = [CM_FIRST + cm_deconnexion];
 virtual void  priseposte  (TMessage & ) = [CM_FIRST + cm_priseposte];
 virtual void  aide        (TMessage & ) = [CM_FIRST + cm_Aide    ];
 virtual void  socket_fin();
 virtual void  socket_close();
 virtual int   socket_client();
};

//**********************************************************************
//                  DEFINITION DES METHODES de TFenetrePrincipale
TFenetrePrincipale::TFenetrePrincipale(PTWindowsObject aparent, LPSTR atitle,
        LPSTR amenu, PTModule pmModule)

 : TWindow(aparent,atitle,pmModule)
{
 Attr.Menu = amenu;   // on transmet le nom du menu (Attr variable de TWindow)
 if(WSAStartup(0x0101,&wasadata) == 0) sock_init=1;
   else sock_init = 0;
 sock_client_cmd = sock_client_rep = INVALID_SOCKET;
 var_connexion=0;
 prise_poste.mot1 = prise_poste.mot2 = 0;
 memset(ip_local,0,sizeof(ip_local));
 dialog_sortie_pl = NULL;
}

//**********************************************************************
// permet de définir l'icone et des caractéristiques
void TFenetrePrincipale::GetWindowClass(WNDCLASS _FAR & awndclass)
{
 TWindow::GetWindowClass(awndclass);
```

```
 awndclass.hIcon = LoadIcon (awndclass.hInstance,"ICONE_CLIENT");
 awndclass.style = CS_HREDRAW | CS_VREDRAW;
}

//***************************************************
// méthode de traitement par défaut des messages
void TFenetrePrincipale::DefWndProc(TMessage & msg)
{
 switch(msg.Message)
   {
    case WM_SYSCOMMAND :   // appel au menu système de la barre de commande
      if((msg.WParam & 0xFFF0) != SC_CLOSE) TWindow::DefWndProc(msg);
       else
        {
         quitter(msg); // demande de destruction avec test
                      // permet de ne pas sortir sans contrôle
         msg.Result = 0;
        }
      break;
    case WM_USER + 1 :
      switch(msg.WParam)
       {
        case TYPE_DEMANDE_VALIDATION :
        if(msg.LParam)
         {
          TRAME_CMD_OS9 trame_env;
          TRAME_REP     trame_rep;
          MESSAGE_A_VALIDER *message_a_valider = (MESSAGE_A_VALIDER *) msg.LParam;
          char *big_trame;
          int  taille;

          big_trame = (char *) malloc(taille = sizeof(TRAME_CMD_OS9)+sizeof(COLIS)*2);
          memset(big_trame,0,taille);

          trame_env.type = TYPE_DEMANDE_VALIDATION;
          memcpy(&trame_env.infos_a_valider,message_a_valider->infos_a_valider, sizeof(INFOS_A_VALIDER));

          memcpy(big_trame,&trame_env,sizeof(TRAME_CMD_OS9));
          if(message_a_valider->infos_a_valider->nb_colis == 1)
            {
             taille -= sizeof(COLIS);
             memcpy(big_trame+sizeof(TRAME_CMD_OS9),message_a_valider->colis1,sizeof(COLIS));
            }
           else if(message_a_valider->infos_a_valider->nb_colis == 2)
            {
             memcpy(big_trame+sizeof(TRAME_CMD_OS9)  ,message_a_valider->colis1,sizeof(COLIS));
             memcpy(big_trame+sizeof(TRAME_CMD_OS9)+sizeof(COLIS),message_a_valider->colis2,
                      sizeof(COLIS));
             }
           else taille =sizeof(TRAME_CMD_OS9);
           send(sock_client_cmd,big_trame,taille,0);
          if(recv(sock_client_rep,(char *) &trame_rep,sizeof(TRAME_REP),0) == sizeof(TRAME_REP))
            msg.Result = trame_rep.err_type + (((long) trame_rep.reponse) << 16);
            else msg.Result = 0x20000l;
          free(big_trame);
         }
         else msg.Result = 0x30000l;
         break;
       case TYPE_DEMANDE_INFOS_PORTILLONS :
        {
         TRAME_CMD_OS9 trame_env, trame_rep;

          memset(&trame_env,0,sizeof(trame_env));
         trame_env.type = TYPE_DEMANDE_INFOS_PORTILLONS;
         send(sock_client_cmd,(char *) &trame_env,sizeof(TRAME_CMD_OS9),0);
         recv(sock_client_rep,(char *) &trame_rep,sizeof(TRAME_CMD_OS9),0);
         if(trame_rep.type == TYPE_REP_INFOS_PORTILLONS)
           PostMessage(dialog_sortie_pl->HWindow,WM_USER+1,TYPE_REP_INFOS_PORTILLONS,
               (LONG) trame_rep.infos_portillons.data);
         break;
        }
       case TYPE_DEMANDE_OUVERTURE  :
       case TYPE_DEMANDE_REINJECTION:
        {
         TRAME_CMD_OS9 trame_env;
         // la réponse est validée par l'ouverture des barrières
```

```
          memset(&trame_env,0,sizeof(TRAME_CMD_OS9));
          trame_env.type = msg.WParam;
          send(sock_client_cmd,(char *) &trame_env,sizeof(TRAME_CMD_OS9),0);
          break;
         }
        default :
           break;
        }
       break;
     case WM_USER :
      {
       switch(WSAGETSELECTEVENT(msg.LParam))
        {
        case FD_CLOSE :
         if(sock_client_cmd == msg.WParam)
            {
            MessageBox(HWindow,
"Le programme va être interrompu:\nAppelez la Maintenance\nAprès réparation relancez le programme",
              "Déconnexion exigée par le serveur",MB_OK);
             socket_fin();
            PostQuitMessage(0);
           }
         break;
        case FD_READ :
         if(sock_client_cmd == msg.WParam)
            {
            TRAME_CMD_OS9 trame_recv;

            if(recv(sock_client_cmd,(char *) &trame_recv,sizeof(trame_recv),0) == sizeof(trame_recv))
              {
               switch(trame_recv.type)
                {
                 case TYPE_ARRIVE_TRACTEUR :
                   if(dialog_sortie_pl)
                      {
                      static char carte[LCARTE];
                      memcpy(carte,trame_recv.carte,LCARTE);
                      PostMessage(dialog_sortie_pl->HWindow,WM_USER+1,TYPE_ARRIVE_TRACTEUR,(LONG) carte);
                      }
                   break;
                 case TYPE_REP_INFOS_PORTILLONS :
                   if(dialog_sortie_pl)
                      {
                      PostMessage(dialog_sortie_pl->HWindow,WM_USER+1,TYPE_REP_INFOS_PORTILLONS,
                        (LONG) trame_recv.infos_portillons.data);
                      }
                   break;
                 default :
                    break;
                }
              }
            }
           break;
          default : break ;
         }
       }
       break;
     default :
       TWindow::DefWndProc(msg);
       break;
   }
}

//****************************************************
// Méthode appelée avant le premier affichage de la fenêtre
void   TFenetrePrincipale::SetupWindow    ()
{
 TWindow::SetupWindow();
 if(sock_init)
   { // remplissage de ip_local et de ip_serv
    // ? ? ? ? ? ? ? ? ? ? ? ? ?
   }
}

//****************************************************
// méthode appelée pour que la fenêtre détruise ses objets avec CTRL sup
// CanClose() est appelé ultérieurement
```

```
void   TFenetrePrincipale::WMClose(RTMessage Msg)
{
 socket_fin();// destruction des objets créés par la fenêtre
 MessageBox(HWindow,"",       "WMCLOSE",MB_OK);
 TWindow::WMClose(Msg);
}

//***************************************************
// CanClose() ne sera pas appelé
void TFenetrePrincipale::quitter        (TMessage &)
{
 if(MessageBox(HWindow,
        "Désirez vous sortir.",
       "Demande de Sortie",MB_YESNO) == IDYES)
   {
    socket_fin();// destruction des objets créés par la fenêtre
    PostQuitMessage(0);
   }
}

//***************************************************
void TFenetrePrincipale::socket_fin()
{
 if(sock_init)
   {
    socket_close();
    sock_init=0;
    while(WSACleanup() == WSANOTINITIALISED);
   }
}

//***************************************************
void TFenetrePrincipale::socket_close()
{
 var_connexion = 0;

 if(sock_init)
   {
    if(sock_client_cmd != INVALID_SOCKET)
      {
       closesocket(sock_client_cmd);
       sock_client_cmd = INVALID_SOCKET;
      }
    if(sock_client_rep != INVALID_SOCKET)
      {
       closesocket(sock_client_rep);
       sock_client_rep = INVALID_SOCKET;
      }
   }
}

//***************************************************
int TFenetrePrincipale::socket_client()
{
// long   hosts_serveur = * ((long *) ip_serv); //inet_addr("4.0.0.90");
 struct sockaddr_in addr_serveur_cmd, addr_serveur_rep;
 TRAME_CMD_OS9 trame_env;
 TRAME_REP     trame_rep;

 if(sock_init)
   {
    memset(&trame_env,0,sizeof(TRAME_CMD_OS9));
    if(GetApplication()->ExecDialog(new TDialog_Login(this,ip_local,ip_serv,&trame_env.login)) == IDOK)
      {
       if(((sock_client_cmd=socket(AF_INET,SOCK_STREAM,0)) != INVALID_SOCKET) &&
         ((sock_client_rep=socket(AF_INET,SOCK_STREAM,0)) != INVALID_SOCKET))
         {
         addr_serveur_cmd.sin_family          = addr_serveur_rep.sin_family           = AF_INET;
         addr_serveur_cmd.sin_addr.S_un.S_addr=addr_serveur_rep.sin_addr.S_un.S_addr=*((long *) ip_serv);
         addr_serveur_cmd.sin_port             = htons(PORT_CMD_OS9);
         addr_serveur_rep.sin_port             = htons(PORT_REP_OS9);
         if( connect(sock_client_cmd,(struct sockaddr *)&addr_serveur_cmd,sizeof(addr_serveur_cmd))
               !=SOCKET_ERROR)
           if(connect(sock_client_rep,(struct sockaddr *)&addr_serveur_rep,sizeof(addr_serveur_rep))
               !=SOCKET_ERROR)
             {
              if(WSAAsyncSelect(sock_client_cmd,HWindow,WM_USER,FD_READ|FD_CLOSE)==0)
```

```
              {
              trame_env.type = TYPE_LOGIN;
              send(sock_client_cmd,(char *) &trame_env,sizeof(TRAME_CMD_OS9),0);
               if(recv(sock_client_rep,(char *) &trame_rep,sizeof(TRAME_REP),0) == sizeof(TRAME_REP))
                 switch(trame_rep.err_type)
                  {
                   case LOGIN_OK :
                     var_connexion = 1;
                      break;
                   case ERR_NOM_INCONNU   :
                     MessageBox(HWindow,"Nom de Login Inconnu","Demmande connexion",MB_OK);
                      break;
                   case ERR_PASSWD:
                     MessageBox(HWindow,"Mot de passe non valide","Demmande connexion",MB_OK);
                     break;
                   default:
                     MessageBox(HWindow,"Réponse serveur incompréhensible","Demmande connexion",MB_OK);
                      break;
                     }
               else MessageBox(HWindow,"Réponse serveur incompréhensible","Demmande connexion",MB_OK);
              }
             else MessageBox(HWindow,"Erreur sur WSAAsyncSelect","Demmande connexion",MB_OK);
            }
           else MessageBox(HWindow,"Serveur (réponse) non disponible " ,"Demmande connexion",MB_OK);
        else    MessageBox(HWindow,"Serveur (commande) non disponible ","Demmande connexion",MB_OK);
        if(!var_connexion) socket_close();
       }
     }
  }
 return(var_connexion);
}

//****************************************************
void TFenetrePrincipale::priseposte    (TMessage &)
{
 TRAME_CMD_OS9 trame_env;
 TRAME_REP     trame_rep;

 if(var_connexion == 1)
   {
    if(GetApplication()->ExecDialog(new TDialog_PrisePoste(this,&prise_poste)) == IDOK)
      {
       memset(&trame_env,0,sizeof(TRAME_CMD_OS9));
       trame_env.type = TYPE_PRISE_POSTE;
       trame_env.prise_poste.mot1=prise_poste.mot1;
       trame_env.prise_poste.mot2=prise_poste.mot2;
       send(sock_client_cmd,(char *) &trame_env,sizeof(TRAME_CMD_OS9),0);
       recv(sock_client_rep,(char *) &trame_rep,sizeof(TRAME_REP),0);
         if(trame_rep.reponse == REP_OK)
           {
            int fin=0;
            switch(prise_poste.mot1)
              {
               case 0: // rien
                 break;
              case 1: // Entrée PL 1
              case 2: // Entrée PL 2
               GetApplication()->ExecDialog(dialog_sortie_pl=new TDialog_Sortie_PL(this,prise_poste.mot1));
                dialog_sortie_pl = NULL;
                fin = 1;
                break;
              case 3: // Sortie PL 1
              case 4: // Sortie PL 2
                GetApplication()->ExecDialog(dialog_sortie_pl = new TDialog_Sortie_PL(this,prise_poste.mot1));
                dialog_sortie_pl = NULL;
                  fin = 1;
                break;
              case 5: // E/S rail (non traité)
              case 6:
              case 7:
              case 8: // E/S convoi (non traité)
                  fin = 1;
                break;
              default: // ERR
                 break;
              }
            if(fin)
              {
```

```
                memset(&trame_env,0,sizeof(trame_env));
                trame_env.type = TYPE_PRISE_POSTE;
                trame_env.prise_poste.mot1=0;
                trame_env.prise_poste.mot2=0;
                send(sock_client_cmd,(char *) &trame_env,sizeof(trame_env),0);
                recv(sock_client_rep,(char *) &trame_rep,sizeof(trame_rep),0);
                if(trame_rep.reponse != REP_OK) MessageBox(HWindow,
                  "Impossible:\n Appelez la maintenance","Demmande connexion",MB_OK);
              }
            }
          else MessageBox(HWindow,"Impossible:\n Appelez la maintenance","Prise Poste",MB_OK);
      }
   }
}

//***************************************************
void TFenetrePrincipale::connexion    (TMessage &msg)
{
 if(var_connexion == 0)
   {
    if(socket_client() == 1)
      {// l'opérateur est reconnu
       DestroyMenu(GetMenu(HWindow));
       SetMenu(HWindow,LoadMenu(HInstance,"MENU_CLIENT_CONNECTE"));
       priseposte(msg);
      }
   }
}

//***************************************************
void TFenetrePrincipale::deconnexion    (TMessage &)
{
 if(var_connexion ==1)
   {
    DestroyMenu(GetMenu(HWindow));
    SetMenu(HWindow,LoadMenu(HInstance,"MENU_CLIENT_DEPART"));
    socket_close();
   }
}

//***************************************************
void TFenetrePrincipale::aide        (TMessage &)
{
 WinHelp(HWindow,"superv.hlp",HELP_CONTENTS,0l);
}

//*********************************************************************
//                   DECLARATION DU TYPE de Tmyapplication
//*********************************************************************
class tmyapplication : public TApplication
{
 protected: // variables
   LPSTR nom_fenetre;    // nom de la fenêtre

 public: // Methodes
              tmyapplication(LPSTR lpszName, HANDLE ahInstance, HANDLE ahPrevInstance,
                     LPSTR lpszCmdLine, intnCmdShow);
 virtual void  InitMainWindow();
};

//*********************************************************************
//                   DEFINITION  DU TYPE de Tmyapplication
// Crée un objet TWINDOW générique sans titre (1ère fenêtre)
tmyapplication::tmyapplication(LPSTR lpszName, HANDLE ahInstance,
            HANDLE ahPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
     :TApplication(lpszName,ahInstance,ahPrevInstance,lpszCmdLine,nCmdShow)
{
 HInstance   = ahInstance; // mémorisation de l'instance en global
 nom_fenetre = lpszName;
}

//***************************************************
void tmyapplication::InitMainWindow()
{
 MainWindow = new TFenetrePrincipale(NULL,nom_fenetre,"MENU_CLIENT_DEPART",NULL);
}
```

```
//#######################################################################
//              Programme Principal
int PASCAL WinMain(HANDLE hInstance,HANDLE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{
 tmyapplication myapp("Poste Supervision Mourpiane",hInstance,hPrevInstance,lpszCmdLine,nCmdShow);

 myapp.Run();
 return(myapp.Status);
}
```

2) Fichier de ressources associé

```
#include "bitmap.rc"

MENU_CLIENT_DEPART MENU
BEGIN
        MENUITEM "&Connexion", 101
        MENUITEM "&Déconnexion", 102, GRAYED
        MENUITEM "&Quitter", 100
        MENUITEM "&Aide", 300
END

MENU_CLIENT_CONNECTE MENU
BEGIN
        MENUITEM "&Connexion", 101, GRAYED
        MENUITEM "&Déconnexion", 102
        MENUITEM "&Prise de poste", 103
        MENUITEM "&Quitter", 100
        MENUITEM "&Aide", 300
END

DIALOG_LOGIN DIALOG 18, 18, 212, 92
STYLE DS_MODALFRAME|WS_POPUP|WS_CAPTION|WS_SYSMENU
CAPTION "MOUREPIANE [LOGIN]"
BEGIN
        EDITTEXT 101, 53, 21, 76, 12, ES_LEFT|WS_CHILD|WS_VISIBLE|WS_BORDER|WS_TABSTOP
        CONTROL "", 100, "EDIT", ES_LEFT|ES_READONLY|WS_CHILD|WS_VISIBLE|WS_BORDER|WS_TABSTOP, 53, 4, 77, 12
        LTEXT "Nom:", -1, 34, 24, 19, 8, WS_CHILD|WS_VISIBLE|WS_GROUP
        LTEXT "Addr. IP poste:", -1, 4, 6, 49, 8, WS_CHILD|WS_VISIBLE|WS_GROUP
        CONTROL "", 102, "EDIT", ES_LEFT|ES_PASSWORD|WS_CHILD|WS_VISIBLE|WS_BORDER|WS_TABSTOP, 53, 40, 23, 12
        LTEXT "Mot de Passe:", -1, 6, 42, 47, 9, WS_CHILD|WS_VISIBLE|WS_GROUP
        DEFPUSHBUTTON "&Validation", 1, 16, 67, 38, 14, WS_CHILD|WS_VISIBLE|WS_TABSTOP
        PUSHBUTTON "&QUITTER", 2, 79, 67, 38, 14, WS_CHILD|WS_VISIBLE|WS_TABSTOP
END

DIALOG_PRISE_DE_POSTE DIALOG 18, 18, 209, 120
STYLE DS_MODALFRAME|WS_POPUP|WS_CAPTION|WS_SYSMENU
CAPTION "PRISE de POSTE - OPERATEUR"
BEGIN
        LTEXT "PRISE de POSTE - MOUREPIANE", -1, 62, 2, 112, 8, WS_CHILD|WS_VISIBLE|WS_GROUP
        LISTBOX 101, 10, 31, 79, 76, LBS_NOTIFY|WS_CHILD|WS_VISIBLE|WS_BORDER|WS_VSCROLL
        LTEXT "VOIES P.L. DISPONIBLES", -1, 11, 20, 85, 8, WS_CHILD|WS_VISIBLE|WS_GROUP
        CONTROL "", 102, "LISTBOX", LBS_NOTIFY|LBS_MULTIPLESEL|WS_CHILD|WS_VISIBLE|WS_BORDER|WS_VSCROLL, 109, 32, 79, 49
        LTEXT "INTERPHONES", -1, 122, 21, 85, 8, WS_CHILD|WS_VISIBLE|WS_GROUP
        PUSHBUTTON "&VALIDER", 1, 101, 89, 39, 14, WS_CHILD|WS_VISIBLE|WS_TABSTOP
        PUSHBUTTON "&QUITTER", 2, 152, 89, 39, 14, WS_CHILD|WS_VISIBLE|WS_TABSTOP
END

DIALOG_SORTIE_PL DIALOG 18, 18, 295, 211
STYLE DS_MODALFRAME|WS_POPUP|WS_CAPTION|WS_SYSMENU
CAPTION "SORTIE P.L."
BEGIN
  CONTROL   "&Ré-injection",131,"BUTTON",BS_PUSHBUTTON|WS_CHILD|WS_VISIBLE|WS_DISABLED|WS_TABSTOP,80, 196, 48, 14
  CONTROL   "&Ouverture"   ,132, "BUTTON",BS_PUSHBUTTON|WS_CHILD|WS_VISIBLE|WS_DISABLED|WS_TABSTOP,132,196,48,14
  PUSHBUTTON "&Quitter"    , 2, 220, 196, 44, 14, WS_CHILD|WS_VISIBLE|WS_TABSTOP
  CONTROL "Convoi", 100, "BUTTON", BS_GROUPBOX|WS_CHILD|WS_VISIBLE|WS_GROUP, 4, 0, 288, 52
  CONTROL "&Vide"   , 101, "BUTTON", BS_AUTORADIOBUTTON|WS_CHILD|WS_VISIBLE|WS_DISABLED|WS_TABSTOP, 8, 12, 30, 12
  CONTROL "&1 Colis", 102, "BUTTON", BS_AUTORADIOBUTTON|WS_CHILD|WS_VISIBLE|WS_DISABLED|WS_TABSTOP, 8, 24, 30, 12
  CONTROL "&2 Colis", 103, "BUTTON", BS_AUTORADIOBUTTON|WS_CHILD|WS_VISIBLE|WS_DISABLED|WS_TABSTOP, 8, 36, 30, 12
  CONTROL "Divers", 110, "button", BS_GROUPBOX|WS_CHILD|WS_VISIBLE, 4, 52, 48, 64
  LTEXT "Appel Portail", -1, 8, 88, 41, 8, WS_CHILD|WS_VISIBLE|WS_GROUP
  CONTROL "Tracteur", 120, "button", BS_GROUPBOX|WS_CHILD|WS_VISIBLE, 56, 52, 52, 64
  CONTROL "", 121, "EDIT", ES_LEFT|WS_CHILD|WS_VISIBLE|WS_DISABLED|WS_BORDER|WS_TABSTOP, 60, 64, 44, 12
  LTEXT "N\260 Carte", -1, 60, 81, 32, 8, WS_CHILD|WS_VISIBLE|WS_GROUP
  CONTROL "", 122, "EDIT", ES_LEFT|ES_READONLY|WS_CHILD|WS_VISIBLE|WS_BORDER|WS_TABSTOP, 60, 92, 44, 12
  CONTROL "Colis N\260 1", 200, "BUTTON", BS_GROUPBOX|WS_CHILD|WS_VISIBLE|WS_GROUP, 112, 52, 88, 140
  CONTROL "Cont.", 201, "BUTTON", BS_AUTORADIOBUTTON|WS_CHILD|WS_VISIBLE|WS_DISABLED|WS_TABSTOP, 116, 64, 28, 12
  CONTROL "Autre", 202, "BUTTON", BS_AUTORADIOBUTTON|WS_CHILD|WS_VISIBLE|WS_DISABLED|WS_TABSTOP, 116, 76, 28, 12
  CONTROL "", 210, "BorShade", BSS_GROUP|WS_CHILD|NOT WS_VISIBLE|WS_GROUP, 156, 60, 36, 36
  CONTROL "Normal" , 211, "BUTTON", BS_AUTORADIOBUTTON|WS_CHILD|WS_VISIBLE|WS_DISABLED|WS_TABSTOP, 160, 61, 34, 12
  CONTROL "Transit", 212, "BUTTON", BS_AUTORADIOBUTTON|WS_CHILD|WS_VISIBLE|WS_DISABLED|WS_TABSTOP, 160, 73, 34, 12
  CONTROL "Douane" , 213, "BUTTON", BS_AUTORADIOBUTTON|WS_CHILD|WS_VISIBLE|WS_DISABLED|WS_TABSTOP, 160, 85, 35, 12
  CONTROL "",220,"COMBOBOX",CBS_DROPDOWN|CBS_SORT|CBS_DISABLENOSCROLL|WS_CHILD|WS_VISIBLE|WS_DISABLED|WS_VSCROLL|...,116,100,80,44
  CONTROL "&Valider", 130, "BUTTON", BS_PUSHBUTTON|WS_CHILD|WS_VISIBLE|WS_DISABLED|WS_TABSTOP, 32, 196, 44, 14
  CONTROL "Vide", 221, "BUTTON", BS_AUTORADIOBUTTON|WS_CHILD|WS_VISIBLE|WS_DISABLED|WS_GROUP|WS_TABSTOP, 116, 112, 28, 12
```

```
   CONTROL"",230,"EDIT", ES_LEFT|ES_MULTILINE|ES_AUTOVSCROLL|WS_CHILD|WS_VISIBLE|WS_DISABLED|WS_BORDER|WS_VSCROLL|…,116,136,80,20
   LTEXT "Commentaire", -1, 116, 128, 68, 8, WS_CHILD|WS_VISIBLE|WS_GROUP
   CONTROL "",240,"EDIT",ES_LEFT|ES_MULTILINE|ES_AUTOVSCROLL|WS_CHILD|WS_VISIBLE|WS_DISABLED|WS_BORDER|WS_VSCROLL|…,116,168,80,20
   LTEXT "Commentaire Manutent.", -1, 116, 160, 80, 8, WS_CHILD|WS_VISIBLE|WS_GROUP
   CONTROL "Colis N\260 2", 300, "BUTTON", BS_GROUPBOX|WS_CHILD|WS_VISIBLE|WS_GROUP, 204, 52, 88, 140
   CONTROL "Cont.", 301, "BUTTON", BS_AUTORADIOBUTTON|WS_CHILD|WS_VISIBLE|WS_DISABLED|WS_TABSTOP, 208, 64, 28, 12
   CONTROL "Autre", 302, "BUTTON", BS_AUTORADIOBUTTON|WS_CHILD|WS_VISIBLE|WS_DISABLED|WS_TABSTOP, 208, 76, 28, 12
   CONTROL "", 310, "BorShade", BSS_GROUP|WS_CHILD|NOT WS_VISIBLE|WS_GROUP, 248, 60, 36, 36
   CONTROL "Normal" , 311, "BUTTON", BS_AUTORADIOBUTTON|WS_CHILD|WS_VISIBLE|WS_DISABLED|WS_TABSTOP, 252, 61, 34, 12
   CONTROL "Transit", 312, "BUTTON", BS_AUTORADIOBUTTON|WS_CHILD|WS_VISIBLE|WS_DISABLED|WS_TABSTOP, 252, 73, 34, 12
   CONTROL "Douane" , 313, "BUTTON", BS_AUTORADIOBUTTON|WS_CHILD|WS_VISIBLE|WS_DISABLED|WS_TABSTOP, 252, 85, 35, 12
   CONTROL »",320,"COMBOBOX",CBS_DROPDOWN|CBS_SORT|CBS_DISABLENOSCROLL|WS_CHILD|WS_VISIBLE|WS_DISABLED|WS_VSCROLL|WS_TABSTOP,208,100,80,44
   CONTROL "Vide", 321, "BUTTON",BS_AUTORADIOBUTTON|WS_CHILD|WS_VISIBLE|WS_DISABLED|WS_GROUP|WS_TABSTOP, 208, 112, 28, 12
   CONTROL "",330,"EDIT",ES_LEFT|ES_MULTILINE|ES_AUTOVSCROLL|WS_CHILD|WS_VISIBLE|WS_DISABLED|WS_BORDER|WS_VSCROLL|WS_TABSTOP,208,136,80,20
   LTEXT "Commentaire", -1, 208, 128, 68, 8, WS_CHILD|WS_VISIBLE|WS_GROUP
   CONTROL "",340,"EDIT",ES_LEFT|ES_MULTILINE|ES_AUTOVSCROLL|WS_CHILD|WS_VISIBLE|WS_DISABLED|WS_BORDER|WS_VSCROLL|WS_TABSTOP,208,168,80,20
   LTEXT "Commentaire Manutent.", -1, 208, 160, 80, 8, WS_CHILD|WS_VISIBLE|WS_GROUP
   CONTROL "Caméra", -1, "button", BS_GROUPBOX|BS_LEFTTEXT|WS_CHILD|WS_VISIBLE, 5, 121, 104, 72
   PUSHBUTTON "Zoom +", 401, 9, 129, 28, 14, WS_CHILD|WS_VISIBLE|WS_TABSTOP
   PUSHBUTTON "Zoom -", 401, 9, 145, 28, 14, WS_CHILD|WS_VISIBLE|WS_TABSTOP
   PUSHBUTTON "Haut",   410, 41, 129, 28, 28, WS_CHILD|WS_VISIBLE|WS_TABSTOP
   PUSHBUTTON "Cnt +",  405, 73, 129, 28, 14, WS_CHILD|WS_VISIBLE|WS_TABSTOP
   PUSHBUTTON "Cnt -",  406, 73, 145, 28, 14, WS_CHILD|WS_VISIBLE|WS_TABSTOP
   PUSHBUTTON "Bas",    411, 41, 161, 28, 28, WS_CHILD|WS_VISIBLE|WS_TABSTOP
   PUSHBUTTON "Gauche", 415, 9, 161, 28, 28, WS_CHILD|WS_VISIBLE|WS_TABSTOP
   PUSHBUTTON "Droite", 416, 73, 161, 28, 28, WS_CHILD|WS_VISIBLE|WS_TABSTOP
END

DIALOG_ENTREE_PL DIALOG 18, 18, 295, 211
STYLE DS_MODALFRAME|WS_POPUP|WS_CAPTION|WS_SYSMENU
CAPTION "ENTREE P.L."
BEGIN
// idem DIALOG_SORTIE_PL sauf que les commentaires manutentionaires n'existent pas
// combobox 240 et 340
// ???????????????????
END

ICON_CLIENT ICON
BEGIN
// ???????????????????
END
```

An Open Interface for Network Programming under Microsoft_ Windows_ Version 1.1 20 January 1993

Martin Hall Mark Towfiq Geoff Arnold David Treadwell Henry Sanders

1) Table d'index :

1) Table d'index :......1
2) accept() : Accept a connection on a socket......1
2) bind() : Associate a local address with a socket......2
4) closesocket() : Close a socket......2
5) connect() : Establish a connection to a peer......2
6) getsockname() : Get the local name for a socket......3
7) listen() : Establish a socket to listen for incoming connection......3
8) recv() : Receive data from a socket......3
9) recvfrom() : Receive a datagram and store the source address......4
10) send() : Send data on a connected socket......4
11) sendto() : Send data to a specific destination......5
12) shutdown() : Disable sends and/or receives on a socket......5
13) socket() : Create a socket......6
14) gethostbyaddr() : Get host information corresponding to an address......6
15) gethostbyname() : Get host information corresponding to a hostname......7
16) gethostname() : Return the standard host name for the local machine......7
17) WSAAsyncSelect() : Request event notification for a socket......7
18) WSACleanup() : Terminate use of the Windows Sockets DLL......9
19) WSAStartup()......10

2) accept() : Accept a connection on a socket.

FAR accept (SOCKET *s*, struct sockaddr FAR * *addr*, int FAR * *addrlen*);

s A descriptor identifying a socket which is listening for connections after a **listen**().

addr An optional pointer to a buffer which receives the address of the connecting entity, as known to the communications layer. The exact format of the *addr* argument is determined by the address family established when the socket was created.

addrlen An optional pointer to an integer which contains the length of the address *addr*.

Remarks

This routine extracts the first connection on the queue of pending connections on *s*, creates a new socket with the same properties as *s* and returns a handle to the new socket. If no pending connections are present on the queue, and the socket is not marked as non-blocking, **accept()** blocks the caller until a connection is present. If the socket is marked non-blocking and no pending connections are present on the queue, **accept()** returns an error as described below. The accepted socket may not be used to accept more connections. The original socket remains open.

The argument *addr* is a result parameter that is filled in with the address of the connecting entity, as known to the communications layer. The exact format of the *addr* parameter is determined by the address family in which the communication is occurring. The *addrlen* is a value-result parameter; it should initially contain the amount of space pointed to by *addr*; on return it will contain the actual length (in bytes) of the address returned. This call is used with connection-based socket types such as SOCK_STREAM. If *addr* and/or *addrlen* are equal to NULL, then no information about the remote address of the accepted socket is returned.

Return Value

If no error occurs, **accept**() returns a value of type SOCKET which is a descriptor for the accepted packet. Otherwise, a value of INVALID_SOCKET is returned, and a specific error code may be retrieved by calling **WSAGetLastError**().

The integer referred to by *addrlen* initially contains the amount of space pointed to by *addr*. On return it will contain the actual length in bytes of the address returned.

2) bind() : Associate a local address with a socket.

int PASCAL FAR bind (SOCKET *s,* const struct sockaddr FAR * *name*, int *namelen*);

s A descriptor identifying an unbound socket.

name The address to assign to the socket. The sockaddr structure is defined as follows:

```
struct sockaddr {
        u_short     sa_family;
        char        sa_data[14];
        };
```

namelen The length of the *name*.

Remarks

This routine is used on an unconnected datagram or stream socket, before subsequent **connect**()s or **listen**()s. When a socket is created with **socket**(), it exists in a name space (address family), but it has no name assigned. **bind**() establishes the local association (host address/port number) of the socket by assigning a local name to an unnamed socket.

In the Internet address family, a name consists of several components. For SOCK_DGRAM and SOCK_STREAM, the name consists of three parts: a host address, the protocol number (set implicitly to UDP or TCP, respectively), and a port number which identifies the application. If an application does not care what address is assigned to it, it may specify an Internet address equal to INADDR_ANY, a port equal to 0, or both. If the Internet address is equal to INADDR_ANY, any appropriate network interface will be used; this simplifies application programming in the presence of multi-homed hosts. If the port is specified as 0, the Windows Sockets implementation will assign a unique port to the application with a value between 1024 and 5000. The application may use **getsockname()** after **bind()** to learn the address that has been assigned to it, but note that **getsockname()** will not necessarily fill in the Internet address until the socket is connected, since several Internet addresses may be valid if the host is multi-homed.

If an application desires to bind to an arbitrary port outside of the range 1024 to 5000, such as the case of rsh which must bind to any reserved port, code similar to the following may be used:

Return Value

If no error occurs, **bind**() returns 0. Otherwise, it returns SOCKET_ERROR, and a specific error code may be retrieved by calling **WSAGetLastError**().

4) closesocket() : Close a socket.

int PASCAL FAR closesocket (SOCKET *s*);

s A descriptor identifying a socket.

Remarks

This function closes a socket. More precisely, it releases the socket descriptor *s*, so that further references to *s* will fail with the error WSAENOTSOCK. If this is the last reference to the underlying socket, the associated naming information and queued data are discarded.

The semantics of **closesocket**() are affected by the socket options SO_LINGER and SO_DONTLINGER as follows:

Option	Interval	Type of close	Wait for close?
SO_DONTLINGER	Don't care	Graceful	No
SO_LINGER	Zero	Hard	No
SO_LINGER	Non-zero	Graceful	Yes

If SO_LINGER is set (i.e. the *l_onoff* field of the linger structure is non-zero; see sections **Erreur! Source du renvoi introuvable.**, **Erreur! Source du renvoi introuvable.** and **Erreur! Source du renvoi introuvable.**) with a zero timeout interval (*l_linger* is zero), **closesocket**() is not blocked even if queued data has not yet been sent or acknowledged. This is called a "hard" or "abortive" close, because the socket's virtual circuit is reset immediately, and any unsent data is lost. Any **recv()** call on the remote side of the circuit will fail with WSAECONNRESET.

If SO_LINGER is set with a non-zero timeout interval, the **closesocket()** call blocks until the remaining data has been sent or until the timeout expires. This is called a graceful disconnect. Note that if the socket is set to non-blocking and SO_LINGER is set to a non-zero timeout, the call to **closesocket()** will fail with an error of WSAEWOULDBLOCK.

If SO_DONTLINGER is set on a stream socket (i.e. the *l_onoff* field of the linger structure is zero; see sections **Erreur! Source du renvoi introuvable.**, **Erreur! Source du renvoi introuvable.** and **Erreur! Source du renvoi introuvable.**), the **closesocket()** call will return immediately. However, any data queued for transmission will be sent if possible before the underlying socket is closed. This is also called a graceful disconnect. Note that in this case the Windows Sockets implementation may not release the socket and other resources for an arbitrary period, which may affect applications which expect to use all available sockets.

Return Value

If no error occurs, **closesocket()** returns 0. Otherwise, a value of SOCKET_ERROR is returned, and a specific error code may be retrieved by calling **WSAGetLastError().**

5) connect() : Establish a connection to a peer.

int PASCAL FAR connect (SOCKET *s*, **const struct sockaddr FAR *** *name*, **int** *namelen* **);**

s A descriptor identifying an unconnected socket.

name The name of the peer to which the socket is to be connected.

namelen The length of the *name*.

Remarks

This function is used to create a connection to the specified foreign association. The parameter *s* specifies an unconnected datagram or stream socket If the socket is unbound, unique values are assigned to the local association by the system, and the socket is marked as bound. Note that if the address field of the *name* structure is all zeroes, **connect()** will return the error WSAEADDRNOTAVAIL.

For stream sockets (type SOCK_STREAM), an active connection is initiated to the foreign host using *name* (an address in the name space of the socket). When the socket call completes successfully, the socket is ready to send/receive data.

For a datagram socket (type SOCK_DGRAM), a default destination is set, which will be used on subsequent **send**() and **recv**() calls.

Return Value

If no error occurs, **connect()** returns 0. Otherwise, it returns SOCKET_ERROR, and a specific error code may be retrieved by calling **WSAGetLastError().**

On a blocking socket, the return value indicates success or failure of the connection attempt.

On a non-blocking socket, if the return value is SOCKET_ERROR an application should call **WSAGetLastError().** If this indicates an error code of WSAEWOULDBLOCK, then your application can either:

1. Use **select()** to determine the completion of the connection request by checking if the socket is writeable, or
2. If your application is using the message-based **WSAAsyncSelect()** to indicate interest in connection events, then your application will receive an FD_CONNECT message when the connect operation is complete.

6) getsockname() : Get the local name for a socket.

int PASCAL FAR getsockname (SOCKET s**, struct sockaddr FAR *** *name***, int FAR *** *namelen* **);**

s A descriptor identifying a bound socket.
name Receives the address (name) of the socket.
namelen The size of the *name* buffer.

Remarks

getsockname() retrieves the current name for the specified socket descriptor in *name*. It is used on a bound and/or connected socket specified by the *s* parameter. The local association is returned. This call is especially useful when a **connect()** call has been made without doing a **bind()** first; this call provides the only means by which you can determine the local association which has been set by the system.

On return, the *namelen* argument contains the actual size of the name returned in bytes.

If a socket was bound to INADDR_ANY, indicating that any of the host's IP addresses should be used for the socket, **getsockname()** will not necessarily return information about the host IP address, unless the socket has been connected with **connect()** or **accept().** A Windows Sockets application must not assume that the IP address will be changed from INADDR_ANY unless the socket is connected. This is because for a multi-homed host the IP address that will be used for the socket is unknown unless the socket is connected.

Return Value

If no error occurs, **getsockname()** returns 0. Otherwise, a value of SOCKET_ERROR is returned, and a specific error code may be retrieved by calling **WSAGetLastError().**

7) listen() : Establish a socket to listen for incoming connection.

int PASCAL FAR listen (SOCKET *s***, int** *backlog* **);**

s A descriptor identifying a bound, unconnected socket.
backlog The maximum length to which the queue of pending connections may grow.

Remarks

To accept connections, a socket is first created with **socket**(), a backlog for incoming connections is specified with **listen**(), and then the connections are accepted with **accept**(). **listen**() applies only to sockets that support connections, i.e. those of type SOCK_STREAM. The socket *s* is put into "passive'' mode where incoming connections are acknowledged and queued pending acceptance by the process.

This function is typically used by servers that could have more than one connection request at a time: if a connection request arrives with the queue full, the client will receive an error with an indication of WSAECONNREFUSED.

listen() attempts to continue to function rationally when there are no available descriptors. It will accept connections until the queue is emptied. If descriptors become available, a later call to **listen**() or **accept**() will re-fill the queue to the current or most recent "backlog'', if possible, and resume listening for incoming connections.

Return Value

If no error occurs, **listen**() returns 0. Otherwise, a value of SOCKET_ERROR is returned, and a specific error code may be retrieved by calling **WSAGetLastError**().

8) recv() : Receive data from a socket.

int PASCAL FAR recv (SOCKET *s***, char FAR *** *buf***, int** *len***, int** *flags* **);**

s A descriptor identifying a connected socket.
buf A buffer for the incoming data.

len The length of *buf*.
flags Specifies the way in which the call is made.

Remarks

This function is used on connected datagram or stream sockets specified by the *s* parameter and is used to read incoming data.

For sockets of type SOCK_STREAM, as much information as is currently available up to the size of the buffer supplied is returned. If the socket has been configured for in-line reception of out-of-band data (socket option SO_OOBINLINE) and out-of-band data is unread, only out-of-band data will be returned. The application may use the **ioctlsocket()** SIOCATMARK to determine whether any more out-of-band data remains to be read.

For datagram sockets, data is extracted from the first enqueued datagram, up to the size of the buffer supplied. If the datagram is larger than the buffer supplied, the buffer is filled with the first part of the datagram, the excess data is lost, and **recv()** returns the error WSAEMSGSIZE.

If no incoming data is available at the socket, the **recv()** call waits for data to arrive unless the socket is non-blocking. In this case a value of SOCKET_ERROR is returned with the error code set to WSAEWOULDBLOCK. The **select()** or **WSAAsyncSelect()** calls may be used to determine when more data arrives.

If the socket is of type SOCK_STREAM and the remote side has shut down the connection gracefully, a **recv()** will complete immediately with 0 bytes received. If the connection has been reset, a **recv()** will fail with the error WSAECONNRESET.

Flags may be used to influence the behavior of the function invocation beyond the options specified for the associated socket. That is, the semantics of this function are determined by the socket options and the *flags* parameter. The latter is constructed by or-ing any of the following values:

Value	Meaning
MSG_PEEK	Peek at the incoming data. The data is copied into the buffer but is not removed from the input queue.
MSG_OOB	Process out-of-band data (See section **Erreur! Source du renvoi introuvable.** for a discussion of this topic.)

Return Value

If no error occurs, **recv()** returns the number of bytes received. If the connection has been closed, it returns 0. Otherwise, a value of SOCKET_ERROR is returned, and a specific error code may be retrieved by calling **WSAGetLastError()**.

9) recvfrom() : Receive a datagram and store the source address.

int PASCAL FAR recvfrom (SOCKET *s*, **char FAR *** *buf*, **int** *len*, **int** *flags*, **struct sockaddr FAR *** *from*, **int FAR *** *fromlen* **);**

s A descriptor identifying a bound socket.
buf A buffer for the incoming data.
len The length of *buf*.
flags Specifies the way in which the call is made.
from An optional pointer to a buffer which will hold the source address upon return.
fromlen An optional pointer to the size of the *from* buffer.

Remarks

This function is used to read incoming data on a (possibly connected) socket and capture the address from which the data was sent.

For sockets of type SOCK_STREAM, as much information as is currently available up to the size of the buffer supplied is returned. If the socket has been configured for in-line reception of out-of-band data (socket option SO_OOBINLINE) and out-of-band data is unread, only out-of-band data will be returned. The application may use the **ioctlsocket()** SIOCATMARK to determine whether any more out-of-band data remains to be read. The *from* and *fromlen* parameters are ignored for SOCK_STREAM sockets.

For datagram sockets, data is extracted from the first enqueued datagram, up to the size of the buffer supplied. If the datagram is larger than the buffer supplied, the buffer is filled with the first part of the message, the excess data is lost, and **recvfrom()** returns the error code WSAEMSGSIZE.

If *from* is non-zero, and the socket is of type SOCK_DGRAM, the network address of the peer which sent the data is copied to the corresponding struct sockaddr. The value pointed to by *fromlen* is initialized to the size of this structure, and is modified on return to indicate the actual size of the address stored there.

If no incoming data is available at the socket, the **recvfrom()** call waits for data to arrive unless the socket is non-blocking. In this case a value of SOCKET_ERROR is returned with the error code set to WSAEWOULDBLOCK. The **select()** or **WSAAsyncSelect()** calls may be used to determine when more data arrives.

If the socket is of type SOCK_STREAM and the remote side has shut down the connection gracefully, a **recvfrom()** will complete immediately with 0 bytes received. If the connection has been reset **recv()** will fail with the error WSAECONNRESET.

Flags may be used to influence the behavior of the function invocation beyond the options specified for the associated socket. That is, the semantics of this function are determined by the socket options and the *flags* parameter. The latter is constructed by or-ing any of the following values:

Value	Meaning
MSG_PEEK	Peek at the incoming data. The data is copied into the buffer but is not removed from the input queue.
MSG_OOB	Process out-of-band data (See section **Erreur! Source du renvoi introuvable.** for a discussion of this topic.)

Return Value

If no error occurs, **recvfrom()** returns the number of bytes received. If the connection has been closed, it returns 0. Otherwise, a value of SOCKET_ERROR is returned, and a specific error code may be retrieved by calling **WSAGetLastError()**.

10) send() : Send data on a connected socket.

int PASCAL FAR send (SOCKET *s*, **const char FAR** * *buf*, **int** *len*, **int** *flags* **);**

s	A descriptor identifying a connected socket.
buf	A buffer containing the data to be transmitted.
len	The length of the data in *buf*.
flags	Specifies the way in which the call is made.

Remarks

send() is used on connected datagram or stream sockets and is used to write outgoing data on a socket. For datagram sockets, care must be taken not to exceed the maximum IP packet size of the underlying subnets, which is given by the *iMaxUdpDg* element in the WSAData structure returned by **WSAStartup()**. If the data is too long to pass atomically through the underlying protocol the error WSAEMSGSIZE is returned, and no data is transmitted.

Note that the successful completion of a **send()** does not indicate that the data was successfully delivered.

If no buffer space is available within the transport system to hold the data to be transmitted, **send**() will block unless the socket has been placed in a non-blocking I/O mode. On non-blocking SOCK_STREAM sockets, the number of bytes written may be between 1 and the requested length, depending on buffer availability on both the local and foreign hosts. The **select**() call may be used to determine when it is possible to send more data.

Flags may be used to influence the behavior of the function invocation beyond the options specified for the associated socket. That is, the semantics of this function are determined by the socket options and the *flags* parameter. The latter is constructed by or-ing any of the following values:

Value	**Meaning**
MSG_DONTROUTE	Specifies that the data should not be subject to routing. A Windows Sockets supplier may choose to ignore this flag; see also the discussion of the SO_DONTROUTE option in section **Erreur! Source du renvoi introuvable.**.
MSG_OOB	Send out-of-band data (SOCK_STREAM only; see also section **Erreur! Source du renvoi introuvable.**)

Return Value

If no error occurs, **send()** returns the total number of characters sent. (Note that this may be less than the number indicated by *len*.) Otherwise, a value of SOCKET_ERROR is returned, and a specific error code may be retrieved by calling **WSAGetLastError()**.

11) sendto() : Send data to a specific destination.

int PASCAL FAR sendto (SOCKET *s*, **const char FAR** * *buf*, **int** *len*, **int** *flags*, **const struct sockaddr FAR** * *to*, **int** *tolen* **);**

s	A descriptor identifying a socket.
buf	A buffer containing the data to be transmitted.
len	The length of the data in *buf*.
flags	Specifies the way in which the call is made.
to	An optional pointer to the address of the target socket.
tolen	The size of the address in *to*.

Remarks

sendto() is used on datagram or stream sockets and is used to write outgoing data on a socket. For datagram sockets, care must be taken not to exceed the maximum IP packet size of the underlying subnets, which is given by the *iMaxUdpDg* element in the WSAData structure returned by **WSAStartup()**. If the data is too long to pass atomically through the underlying protocol the error WSAEMSGSIZE is returned, and no data is transmitted.

Note that the successful completion of a **sendto()** does not indicate that the data was successfully delivered.

sendto() is normally used on a SOCK_DGRAM socket to send a datagram to a specific peer socket identified by the *to* parameter. On a SOCK_STREAM socket, the *to* and *tolen* parameters are ignored; in this case the **sendto()** is equivalent to **send**().

To send a broadcast (on a SOCK_DGRAM only), the address in the *to* parameter should be constructed using the special IP address INADDR_BROADCAST (defined in **winsock.h**) together with the intended port number. It is generally inadvisable for a broadcast datagram to exceed the size at which fragmentation may occur, which implies that the data portion of the datagram (excluding headers) should not exceed 512 bytes.

If no buffer space is available within the transport system to hold the data to be transmitted, **sendto**() will block unless the socket has been placed in a non-blocking I/O mode. On non-blocking SOCK_STREAM sockets, the number of bytes written may be between 1 and the requested length, depending on buffer availability on both the local and foreign hosts. The **select**() call may be used to determine when it is possible to send more data.

Flags may be used to influence the behavior of the function invocation beyond the options specified for the associated socket. That is, the semantics of this function are determined by the socket options and the *flags* parameter. The latter is constructed by or-ing any of the following values:

Value	**Meaning**
MSG_DONTROUTE	Specifies that the data should not be subject to routing. A Windows Sockets supplier may choose to ignore this flag; see also the discussion of the SO_DONTROUTE option in section **Erreur! Source du renvoi introuvable.**.
MSG_OOB	Send out-of-band data (SOCK_STREAM only; see also section **Erreur! Source du renvoi introuvable.**)

Return Value

If no error occurs, **sendto()** returns the total number of characters sent. (Note that this may be less than the number indicated by *len*.) Otherwise, a value of SOCKET_ERROR is returned, and a specific error code may be retrieved by calling **WSAGetLastError()**.

12) shutdown() : Disable sends and/or receives on a socket.

int PASCAL FAR shutdown (SOCKET *s*, **int** *how* **);**

s	A descriptor identifying a socket.
how	A flag that describes what types of operation will no longer be allowed.

Remarks

shutdown() is used on all types of sockets to disable reception, transmission, or both.

If *how* is 0, subsequent receives on the socket will be disallowed. This has no effect on the lower protocol layers. For TCP, the TCP window is not changed and incoming data will be accepted (but not acknowledged) until the window is exhausted. For UDP, incoming datagrams are accepted and queued. In no case will an ICMP error packet be generated.

If *how* is 1, subsequent sends are disallowed. For TCP sockets, a FIN will be sent.

Setting *how* to 2 disables both sends and receives as described above.

Note that **shutdown()** does not close the socket, and resources attached to the socket will not be freed until **closesocket()** is invoked.

Comments

shutdown() does not block regardless of the SO_LINGER setting on the socket.

An application should not rely on being able to re-use a socket after it has been shut down. In particular, a Windows Sockets implementation is not required to support the use of **connect()** on such a socket.

Return Value

If no error occurs, **shutdown()** returns 0. Otherwise, a value of SOCKET_ERROR is returned, and a specific error code may be retrieved by calling **WSAGetLastError().**

13) socket() : Create a socket.

SOCKET PASCAL FAR socket (int *af*, **int** *type*, **int** *protocol* **);**

af	An address format specification. The only format currently supported is PF_INET, which is the ARPA Internet address format.
type	A type specification for the new socket.
protocol	A particular protocol to be used with the socket, or 0 if the caller does not wish to specify a protocol.

Remarks

socket() allocates a socket descriptor of the specified address family, data type and protocol, as well as related resources. If a protocol is not specified (i.e. equal to 0), the default for the specified connection mode is used.

Only a single protocol exists to support a particular socket type using a given address format. However, the address family may be given as AF_UNSPEC (unspecified), in which case the *protocol* parameter must be specified. The protocol number to use is particular to the "communication domain" in which communication is to take place.

The following *type* specifications are supported:

Type	Explanation
SOCK_STREAM	Provides sequenced, reliable, two-way, connection-based byte streams with an out-of-band data transmission mechanism. Uses TCP for the Internet address family.
SOCK_DGRAM	Supports datagrams, which are connectionless, unreliable buffers of a fixed (typically small) maximum length. Uses UDP for the Internet address family.

Sockets of type SOCK_STREAM are full-duplex byte streams. A stream socket must be in a connected state before any data may be sent or received on it. A connection to another socket is created with a **connect**() call. Once connected, data may be transferred using **send**() and **recv**() calls. When a session has been completed, a **closesocket**() must be performed. Out-of-band data may also be transmitted as described in **send**() and received as described in **recv**().

The communications protocols used to implement a SOCK_STREAM ensure that data is not lost or duplicated. If data for which the peer protocol has buffer space cannot be successfully transmitted within a reasonable length of time, the connection is considered broken and subsequent calls will fail with the error code set to WSAETIMEDOUT.

SOCK_DGRAM sockets allow sending and receiving of datagrams to and from arbitrary peers using **sendto()** and **recvfrom()**. If such a socket is **connect**()ed to a specific peer, datagrams may be send to that peer **send**() and may be received from (only) this peer using **recv**().

Return Value

If no error occurs, **socket()** returns a descriptor referencing the new socket. Otherwise, a value of INVALID_SOCKET is returned, and a specific error code may be retrieved by calling **WSAGetLastError().**

14) gethostbyaddr() : Get host information corresponding to an address.

struct hostent FAR * PASCAL FAR gethostbyaddr (const char FAR * *addr*, **int** *len*, **int** *type* **);**

addr	A pointer to an address in network byte order.
len	The length of the address, which must be 4 for PF_INET addresses.
type	The type of the address, which must be PF_INET.

Remarks

gethostbyaddr() returns a pointer to the following structure which contains the name(s) and address which correspond to the given address.

```
struct hostent {
        char FAR *              h_name;
        char FAR * FAR *        h_aliases;
```

```
        short                   h_addrtype;
        short                   h_length;
        char FAR * FAR * h_addr_list;
};
```

The members of this structure are:

Element	Usage
h_name	Official name of the host (PC).
h_aliases	A NULL-terminated array of alternate names.
h_addrtype	The type of address being returned; for Windows Sockets this is always PF_INET.
h_length	The length, in bytes, of each address; for PF_INET, this is always 4.
h_addr_list	A NULL-terminated list of addresses for the host. Addresses are returned in network byte order.

The macro h_addr is defined to be h_addr_list[0] for compatibility with older software.

The pointer which is returned points to a structure which is allocated by the Windows Sockets implementation. The application must never attempt to modify this structure or to free any of its components. Furthermore, only one copy of this structure is allocated per thread, and so the application should copy any information which it needs before issuing any other Windows Sockets API calls.

Return Value

If no error occurs, **gethostbyaddr()** returns a pointer to the hostent structure described above. Otherwise it returns a NULL pointer and a specific error number may be retrieved by calling **WSAGetLastError().**

15) gethostbyname() : Get host information corresponding to a hostname.

struct hostent FAR * PASCAL FAR gethostbyname (const char FAR * *name* **);**

name A pointer to the name of the host.

Remarks

gethostbyname() returns a pointer to a hostent structure as described under **gethostbyaddr**(). The contents of this structure correspond to the hostname *name*.

The pointer which is returned points to a structure which is allocated by the Windows Sockets implementation. The application must never attempt to modify this structure or to free any of its components. Furthermore, only one copy of this structure is allocated per thread, and so the application should copy any information which it needs before issuing any other Windows Sockets API calls.

A **gethostbyname()** implementation must not resolve IP address strings passed to it. Such a request should be treated exactly as if an unknown host name were passed. An application with an IP address string to resolve should use **inet_addr()** to convert the string to an IP address, then **gethostbyaddr()** to obtain the hostent structure.

Return Value

If no error occurs, **gethostbyname()** returns a pointer to the hostent structure described above. Otherwise it returns a NULL pointer and a specific error number may be retrieved by calling **WSAGetLastError()**.

16) gethostname() : Return the standard host name for the local machine.

int PASCAL FAR gethostname (char FAR * *name*, **int** *namelen* **);**

name A pointer to a buffer that will receive the host name.
namelen The length of the buffer.

Remarks

This routine returns the name of the local host into the buffer specified by the *name* parameter. The host name is returned as a null-terminated string. The form of the host name is dependent on the Windows Sockets implementation--it may be a simple host name, or it may be a fully qualified domain name. However, it is guaranteed that the name returned will be successfully parsed by **gethostbyname()** and **WSAAsyncGetHostByName().**

Return Value

If no error occurs, **gethostname()** returns 0, otherwise it returns SOCKET_ERROR and a specific error code may be retrieved by calling **WSAGetLastError()**.

17) WSAAsyncSelect() : Request event notification for a socket.

int PASCAL FAR WSAAsyncSelect (SOCKET *s*, **HWND** *hWnd*, **unsigned int** *wMsg*, **long** *lEvent* **);**

s A descriptor identifying the socket for which event notification is required.
hWnd A handle identifying the window which should receive a message when a network event occurs.
wMsg The message to be received when a network event occurs.
lEvent A bitmask which specifies a combination of network events in which the application is interested.

Remarks

This function is used to request that the Windows Sockets DLL should send a message to the window *hWnd* whenever it detects any of the network events specified by the *lEvent* parameter. The message which should be sent is specified by the *wMsg* parameter. The socket for which notification is required is identified by *s*.

This function automatically sets socket *s* to non-blocking mode.

The *lEvent* parameter is constructed by or'ing any of the values specified in the following list.

Value	Meaning
FD_READ	Want to receive notification of readiness for reading

FD_WRITE	Want to receive notification of readiness for writing
FD_OOB	Want to receive notification of the arrival of out-of-band data
FD_ACCEPT	Want to receive notification of incoming connections
FD_CONNECT	Want to receive notification of completed connection
FD_CLOSE	Want to receive notification of socket closure

Issuing a **WSAAsyncSelect()** for a socket cancels any previous **WSAAsyncSelect()** for the same socket. For example, to receive notification for both reading and writing, the application must call **WSAAsyncSelect()** with both FD_READ and FD_WRITE, as follows:

```
rc = WSAAsyncSelect(s, hWnd, wMsg, FD_READ|FD_WRITE);
```

It is not possible to specify different messages for different events. The following code will not work; the second call will cancel the effects of the first, and only FD_WRITE events will be reported with message wMsg2:

```
rc = WSAAsyncSelect(s, hWnd, wMsg1, FD_READ);
rc = WSAAsyncSelect(s, hWnd, wMsg2, FD_WRITE);
```

To cancel all notification − i.e., to indicate that the Windows Sockets implementation should send no further messages related to network events on the socket − *lEvent* should be set to zero.

```
rc = WSAAsyncSelect(s, hWnd, 0, 0);
```

Although in this instance **WSAAsyncSelect()** immediately disables event message posting for the socket, it is possible that messages may be waiting in the application's message queue. The application must therefore be prepared to receive network event messages even after cancellation. Closing a socket with **closesocket()** also cancels **WSAAsyncSelect()** message sending, but the same caveat about messages in the queue prior to the **closesocket()** still applies.

Since an **accept()**'ed socket has the same properties as the listening socket used to accept it, any **WSAAsyncSelect()** events set for the listening socket apply to the accepted socket. For example, if a listening socket has **WSAAsyncSelect()** events FD_ACCEPT, FD_READ, and FD_WRITE, then any socket accepted on that listening socket will also have FD_ACCEPT, FD_READ, and FD_WRITE events with the same *wMsg* value used for messages. If a different *wMsg* or events are desired, the application should call **WSAAsyncSelect()**, passing the accepted socket and the desired new information. There is a timing window between the **accept()** call and the call to **WSAAsyncSelect()** to change the events or *wMsg*. An application which desires a different *wMsg* for the listening and **accept()**'ed sockets should ask for only FD_ACCEPT events on the listening socket, then set appropriate events after the **accept()**. Since FD_ACCEPT is never sent for a connected socket and FD_READ, FD_WRITE, FD_OOB, and FD_CLOSE are never sent for listening sockets, this will not impose difficulties.

When one of the nominated network events occurs on the specified socket *s*, the application's window *hWnd* receives message *wMsg*. The *wParam* argument identifies the socket on which a network event has occurred. The low word of *lParam* specifies the network event that has occurred. The high word of *lParam* contains any error code. The error code be any error as defined in **winsock.h**.

The error and event codes may be extracted from the *lParam* using the macros WSAGETSELECTERROR and WSAGETSELECTEVENT, defined in **winsock.h** as:

```
#define WSAGETSELECTERROR(lParam)        HIWORD(lParam)
#define WSAGETSELECTEVENT(lParam)        LOWORD(lParam)
```

The use of these macros will maximize the portability of the source code for the application.

The possible network event codes which may be returned are as follows:

Value	**Meaning**
FD_READ	Socket *s* ready for reading
FD_WRITE	Socket *s* ready for writing
FD_OOB	Out-of-band data ready for reading on socket *s*.
FD_ACCEPT	Socket *s* ready for accepting a new incoming connection
FD_CONNECT	Connection on socket *s* completed
FD_CLOSE	Connection identified by socket *s* has been closed

Return Value

The return value is 0 if the application's declaration of interest in the network event set was successful. Otherwise the value SOCKET_ERROR is returned, and a specific error number may be retrieved by calling **WSAGetLastError()**.

Comments

Although **WSAAsyncSelect()** can be called with interest in multiple events, the application window will receive a single message for each network event.

As in the case of the **select()** function, **WSAAsyncSelect()** will frequently be used to determine when a data transfer operation (**send()** or **recv()**) can be issued with the expectation of immediate success. Nevertheless, a robust application must be prepared for the possibility that it may receive a message and issue a Windows Sockets API call which returns WSAEWOULDBLOCK immediately. For example, the following sequence of events is possible:

- (i) data arrives on socket **s**; Windows Sockets posts **WSAAsyncSelect** message
- (ii) application processes some other message
- (iii) while processing, application issues an **ioctlsocket(s, FIONREAD...)** and notices that there is data ready to be read
- (iv) application issues a **recv(s,...)** to read the data
- (v) application loops to process next message, eventually reaching the **WSAAsyncSelect** message indicating that data is ready to read
- (vi) application issues **recv(s,...)**, which fails with the error WSAEWOULDBLOCK.

Other sequences are possible.

The Windows Sockets DLL will not continually flood an application with messages for a particular network event. Having successfully posted notification of a particular event to an application window, no further message(s) for that network event will be

posted to the application window until the application makes the function call which implicitly reenables notification of that network event.

Event	Re-enabling function
FD_READ	**recv()** or **recvfrom()**
FD_WRITE	**send()** or **sendto()**
FD_OOB	**recv()**
FD_ACCEPT	**accept()**
FD_CONNECT	NONE
FD_CLOSE	NONE

Any call to the reenabling routine, even one which fails, results in reenabling of message posting for the relevant event.

For FD_READ, FD_OOB, and FD_ACCEPT events, message posting is "level-triggered."This means that if the reenabling routine is called and the relevant event is still valid after the call, a **WSAAsyncSelect()** message is posted to the application. This allows an application to be event-driven and not concern itself with the amount of data that arrives at any one time. Consider the following sequence:

(i) Windows Sockets DLL receives 100 bytes of data on socket **s** and posts an FD_READ message.
(ii) The application issues **recv(s, buffptr, 50, 0)** to read 50 bytes.
(iii) The Windows Sockets DLL posts another FD_READ message since there is still data to be read.

With these semantics, an application need not read all available data in response to an FD_READ message--a single **recv()** in response to each FD_READ message is appropriate. If an application issues multiple **recv()** calls in response to a single FD_READ, it may receive multiple FD_READ messages. Such an application may wish to disable FD_READ messages before starting the **recv()** calls by calling **WSAAsyncSelect()** with the FD_READ event not set.

If an event is true when the application initially calls **WSAAsyncSelect()** or when the reenabling function is called, then a message is posted as appropriate. For example, if an application calls **listen()**, a connect attempt is made, then the application calls **WSAAsyncSelect()** specifying that it wants to receive FD_ACCEPT messages for the socket, the Windows Sockets implementation posts an FD_ACCEPT message immediately.

The FD_WRITE event is handled slightly differently. An FD_WRITE message is posted when a socket is first connected with **connect()** or accepted with **accept()**, and then after a **send()** or **sendto()** fails with WSAEWOULDBLOCK and buffer space becomes available. Therefore, an application can assume that sends are possible starting from the first FD_WRITE message and lasting until a send returns WSAEWOULDBLOCK. After such a failure the application will be notified that sends are again possible with an FD_WRITE message.

The FD_OOB event is used only when a socket is configured to receive out-of-band data separately. If the socket is configured to receive out-of-band data in-line, the out-of-band (expedited) data is treated as normal data and the application should register an interest in, and will receive, FD_READ events, not FD_OOB events. An application may set or inspect the way in which out-of-band data is to be handled by using **setsockopt()** or **getsockopt()** for the SO_OOBINLINE option.

The error code in an FD_CLOSE message indicates whether the socket close was graceful or abortive. If the error code is 0, then the close was graceful; if the error code is WSAECONNRESET, then the socket's virtual socket was reset. This only applies to sockets of type SOCK_STREAM.

The FD_CLOSE message is posted when a close indication is received for the virtual circuit corresponding to the socket. In TCP terms, this means that the FD_CLOSE is posted when the connection goes into the FIN WAIT or CLOSE WAIT states. This results from the remote end performing a **shutdown()** on the send side or a **closesocket().**

Please note your application will receive ONLY an FD_CLOSE message to indicate closure of a virtual circuit. It will NOT receive an FD_READ message to indicate this condition.

Notes For Windows Sockets Suppliers

It is the responsibility of the Windows Sockets Supplier to ensure that messages are successfully posted to the application. If a **PostMessage()** operation fails, the Windows Sockets implementation MUST re-post that message as long as the window exists.

Windows Sockets suppliers should use the WSAMAKESELECTREPLY macro when constructing the *lParam* in the message.

When a socket is closed, the Windows Sockets Supplier should purge any messages remaining for posting to the application window. However the application must be prepared to receive, and discard, any messages which may have been posted prior to the **closesocket()**.

18) WSACleanup() : Terminate use of the Windows Sockets DLL.

int PASCAL FAR WSACleanup (void);

Remarks

An application or DLL is required to perform a (successful) **WSAStartup()** call before it can use Windows Sockets services. When it has completed the use of Windows Sockets, the application or DLL must call **WSACleanup()** to deregister itself from a Windows Sockets implementation and allow the implementation to free any resources allocated on behalf of the application or DLL. Any open SOCK_STREAM sockets that are connected when **WSACleanup()** is called are reset; sockets which have been closed with **closesocket()** but which still have pending data to be sent are not affected--the pending data is still sent.

There must be a call to **WSACleanup()** for every call to **WSAStartup()** made by a task. Only the final **WSACleanup()** for that task does the actual cleanup; the preceding calls simply decrement an internal reference count in the Windows Sockets DLL. A naive application may ensure that **WSACleanup()** was called enough times by calling **WSACleanup()** in a loop until it returns WSANOTINITIALISED.

Return Value

The return value is 0 if the operation was successful. Otherwise the value SOCKET_ERROR is returned, and a specific error number may be retrieved by calling **WSAGetLastError()**.

Comments

Attempting to call **WSACleanup()** from within a blocking hook and then failing to check the return code is a common Windows Sockets programming error. If an application needs to quit while a blocking call is outstanding, the application must first cancel the blocking call with **WSACancelBlockingCall()** then issue the **WSACleanup()** call once control has been returned to the application.

Notes For Windows Sockets Suppliers

Well-behaved Windows Sockets applications will make a **WSACleanup()** call to indicate deregistration from a Windows Sockets implementation. This function can thus, for example, be utilized to free up resources allocated to the specific application.

A Windows Sockets implementation must be prepared to deal with an application which terminates without invoking **WSACleanup()** - for example, as a result of an error.

In a multithreaded environment, **WSACleanup()** terminates Windows Sockets operations for all threads.

A Windows Sockets implementation must ensure that **WSACleanup()** leaves things in a state in which the application can invoke **WSAStartup()** to re-establish Windows Sockets usage.

19) WSAStartup()

int PASCAL FAR WSAStartup (WORD *wVersionRequested*, **LPWSADATA** *lpWSAData* **);**

wVersionRequested The highest version of Windows Sockets API support that the caller can use. The high order byte specifies the minor version (revision) number; the low-order byte specifies the major version number.

lpWSAData A pointer to the **WSADATA** data structure that is to receive details of the Windows Sockets implementation.

Remarks

This function **MUST** be the first Windows Sockets function called by an application or DLL. It allows an application or DLL to specify the version of Windows Sockets API required and to retrieve details of the specific Windows Sockets implementation. The application or DLL may only issue further Windows Sockets API functions after a successful **WSAStartup()** invocation.

In order to support future Windows Sockets implementations and applications which may have functionality differences from Windows Sockets 1.1, a negotiation takes place in **WSAStartup()**. The caller of **WSAStartup()** and the Windows Sockets DLL indicate to each other the highest version that they can support, and each confirms that the other's highest version is acceptable. Upon entry to **WSAStartup()**, the Windows Sockets DLL examines the version requested by the application. If this version is higher than the lowest version supported by the DLL, the call succeeds and the DLL returns in *wHighVersion* the highest version it supports and in *wVersion* the minimum of its high version and *wVersionRequested.* The Windows Sockets DLL then assumes that the application will use *wVersion.* If the *wVersion* field of the **WSADATA** structure is unacceptable to the caller, it should call **WSACleanup()** and either search for another Windows Sockets DLL or fail to initialize.

This negotiation allows both a Windows Sockets DLL and a Windows Sockets application to support a range of Windows Sockets versions. An application can successfully utilize a Windows Sockets DLL if there is any overlap in the version ranges. The following chart gives examples of how **WSAStartup()** works in conjunction with different application and Windows Sockets DLL versions:

App versions	DLL Versions	*wVersionRequested*	*wVersion*	*wHighVersion*	End Result
1.1	1.1	1.1	1.1	1.1	use 1.1
1.0 1.1	1.0	1.1	1.0	1.0	use 1.0
1.0	1.0 1.1	1.0	1.0	1.1	use 1.0
1.1	1.0 1.1	1.1	1.1	1.1	use 1.1
1.1	1.0	1.1	1.0	1.0	Application fails
1.0	1.1	1.0	---	---	WSAVERNOTSUPPORTED
1.0 1.1	1.0 1.1	1.1	1.1	1.1	use 1.1
1.1 2.0	1.1	2.0	1.1	1.1	use 1.1
2.0	1.1	2.0	1.1	1.1	Application fails

Once an application or DLL has made a successful **WSAStartup()** call, it may proceed to make other Windows Sockets API calls as needed. When it has finished using the services of the Windows Sockets DLL, the application or DLL must call **WSACleanup()** in order to allow the Windows Sockets DLL to free any resources for the application.

Details of the actual Windows Sockets implementation are described in the WSAData structure defined as follows:

```
struct WSAData {
        WORD            wVersion;
        WORD            wHighVersion;
        char            szDescription[WSADESCRIPTION_LEN+1];
        char            szSystemStatus[WSASYSSTATUS_LEN+1];
        unsigned short  iMaxSockets;
        unsigned short  iMaxUdpDg;
        char FAR *      lpVendorInfo;
        };
```

The members of this structure are:

Element	**Usage**
wVersion	The version of the Windows Sockets specification that the Windows Sockets DLL expects the caller to use.
wHighVersion	The highest version of the Windows Sockets specification that this DLL can support (also encoded as above). Normally this will be the same as *wVersion*.
szDescription	A null-terminated ASCII string into which the Windows Sockets DLL copies a description of the Windows Sockets implementation, including vendor identification. The text (up to 256 characters in length) may contain any characters, but vendors are cautioned against including control and formatting characters: the most likely use that an application will put this to is to display it (possibly truncated) in a status message.

szSystemStatus A null-terminated ASCII string into which the Windows Sockets DLL copies relevant status or configuration information. The Windows Sockets DLL should use this field only if the information might be useful to the user or support staff: it should not be considered as an extension of the *szDescription* field.

iMaxSockets The maximum number of sockets which a single process can potentially open. A Windows Sockets implementation may provide a global pool of sockets for allocation to any process; alternatively it may allocate per-process resources for sockets. The number may well reflect the way in which the Windows Sockets DLL or the networking software was configured. Application writers may use this number as a crude indication of whether the Windows Sockets implementation is usable by the application. For example, an X Windows server might check *iMaxSockets* when first started: if it is less than 8, the application would display an error message instructing the user to reconfigure the networking software. (This is a situation in which the *szSystemStatus* text might be used.) Obviously there is no guarantee that a particular application can actually allocate *iMaxSockets* sockets, since there may be other Windows Sockets applications in use.

iMaxUdpDg The size in bytes of the largest UDP datagram that can be sent or received by a Windows Sockets application. If the implementation imposes no limit, *iMaxUdpDg* is zero. In many implementations of Berkeley sockets, there is an implicit limit of 8192 bytes on UDP datagrams (which are fragmented if necessary). A Windows Sockets implementation may impose a limit based, for instance, on the allocation of fragment reassembly buffers. The minimum value of *iMaxUdpDg* for a compliant Windows Sockets implementation is 512. Note that regardless of the value of *iMaxUdpDg*, it is inadvisable to attempt to send a broadcast datagram which is larger than the Maximum Transmission Unit (MTU) for the network. (The Windows Sockets API does not provide a mechanism to discover the MTU, but it must be no less than 512 bytes.)

lpVendorInfo A far pointer to a vendor-specific data structure. The definition of this structure (if supplied) is beyond the scope of this specification.

An application or DLL may call **WSAStartup()** more than once if it needs to obtain the WSAData structure information more than once. However, the *wVersionRequired* parameter is assumed to be the same on all calls to **WSAStartup()**; that is, an application or DLL cannot change the version of Windows Sockets it expects after the initial call to **WSAStartup().**

There must be one **WSACleanup()** call corresponding to every **WSAStartup()** call to allow third-party DLLs to make use of a Windows Sockets DLL on behalf of an application. This means, for example, that if an application calls **WSAStartup()** three times, it must call **WSACleanup()** three times. The first two calls to **WSACleanup()** do nothing except decrement an internal counter; the final **WSACleanup()** call for the task does all necessary resource deallocation for the task.

Return Value

WSAStartup() returns zero if successful. Otherwise it returns one of the error codes listed below. Note that the normal mechanism whereby the application calls **WSAGetLastError()** to determine the error code cannot be used, since the Windows Sockets DLL may not have established the client data area where the "last error" information is stored.

Notes For Windows Sockets Suppliers

Each Windows Sockets application MUST make a **WSAStartup()** call before issuing any other Windows Sockets API calls. This function can thus be utilized for initialization purposes. Further issues are discussed in the notes for **WSACleanup()**.

THE STANDARD LIBRARY OS-9 System

1) Table d'index

1) Table d'index **1**
2) create() **1**
3) _ev_creat() **1**
4) _ev_link() **1**
5) _ev_wait() **1**
6) _ev_signal() **2**
7) _mkdata_module() **2**
8) modlink() **2**
9) munlink() **2**
10) event.h **2**
11) mode.h **3**
12) module.h (extrait) **3**
13) stdio.h (extrait) **4**

2) create()

#include <modes.h>
int **create**(char * **name,** short **mode,** short **perm [,**int **initial size])**

DESCRIPTION

create() returns a path number to a new file with a name specified by the string pointed to by **name**. The file is available for writing. The access permissions are given by the argument **mode**. The file permission attributes are given in **perm**. The owner of the file is the task owner.

If the file already exists or any other error occurs, -1 is returned and the appropriate error code is placed in the global variable ertno. The valid mode and permission values are available in the <modes.h> header file.

The **initial size** value can be given to indicate the desired initial size for the file. For disk files, the file is pre-extended to this size. For pipe files, the pipe buffer is set to this value. The initial size parameter is used only if the S_ISIZE bit is set in mode.

CAVEAT: This function is similar to the creat() call except it allows the caller to give the exact file attributes desired and does not truncate the file if it is already present.

Directories can not be created with this call. Instead, use the mknod function.

3) _ev_creat()

#include <events.h>
int **_ev_creat**(int **ev value,** int **wait inc,** int **signal tnc,** char * **ev_name)**

DESCRIPTION

This function creates an event. **ev_name** is a pointer to a string containing the name of the event. **ev_value** is the initial value for the event. **wait_inc** and **signal_inc** are increments applied to the event each time the event occurs or is waited for. An event id number is returned if the event is successfully created. -1 is returned if an error occurs and the appropriate error code is placed in the global variable **errno**.

4) _ev_link()

#include <events.h>
int **_ev_link**(char * **ev_name)**

DESCRIPTION

This function links to an existing event. **ev_name** is a pointer to a string containing the name of the event. An event id number is returned if the event is successfully linked. -1 is returned if an error occurs and the appropriate error code is placed in the global variable **errno**.

5) _ev_wait()

#include <events.h>
int **_ev_wait(**int **ev_id,** int **ev_min,** int **ev_max)**

DESCRIPTION

This function waits for an event to occur. The id of the event desired is given in **ev_id**. The event value is compared to the range values given by **ev_min** and **ev_max**. If the event value is not in the specified range, the process will wait until some other process places the value

within the range. Once in range, the wait increment is applied to the event value. The actual event value is returned as the value of the function. -1 is returned if an error occurs and the appropriate error code is placed in the global variable errno.

6) _ev_signal()

```
#include <events . h>
int _ev_signal(int ev_id, short allflag)
```

DESCRIPTION

This function indicates that an event has occurred. **ev_id** is the event id returned from _ev_creat() or _ev_link(). The current event variable is updated by the signal increment (given when the event was created). If **allflag** is zero, the first process waiting for the event is activated. If **allflag** is Ox8000, all processes waiting for the event that have a value in range are activated. -1 is returned if an error occurs and the appropriate error code is placed in the global variable errno.

7) _mkdata_module()

```
#include <module.h>
char * _mkdata_module(char * name, unsigned size, short attr, short perm)
```

DESCRIPTION

This function creates a data memory module. The data module can then be accessed by other processes on the system by modlink(), **name** is the desired name of the module, **size** is the size in bytes of the module, **attr** is the attribut et revision word and **perm** is the access permission word. The memory in the data module is initially cleared to zeroes.

_mkdata_module() returns a pointer to the beginning of the module header...

If the data module can not be created, a value of -1 is returned and the appropriate error code is placed in the global variable errno.

NOTE: The size value does not include the module header and CRC bytes. The size given is the amount of memory available for actual use.

8) modlink()

```
#include <module.h>
mod_exec modlink (char * modname, short typelang)
```

DESCRIPTION

This function searches the module directory for a module with the same name as that pointed by **modname** , and links to it, provided that **typelang** match respective value of type/language in the module. If the module is found, the link count for the module increments by one. You can specify zero to indicate any type or language.

If the link is successful, **modlink()** returns a pointer to the module. If the link fails, it returns -1 and the appropriate error code is placed in the global variable errno.

9) munlink()

```
#include <module.h>
int munlink (mod_exec * module)
```

DESCRIPTION

This function informs the system that the module to which **module** points is no longer required by the process. The module's link count decrements and the module is removed from the memory if its link count reaches zero. **module** must have been a pointer returned by moadlink() or moadload().

munlink() returns -1 on error and the appropriate error code is placed in the global variable errno.

10) event.h

```
/*   Events.h      OS-9 Event and Semaphores  */

#define EV_SIZE     32                /* event entry size */
#define EV_ALLPROCS 0x8000            /* activate all processes in range */

typedef struct _evstr {
   unsigned short _ev_eid;    /* event id number */
   char _ev_name[12];         /* event name */
   int _ev_value;             /* current event value */
   short _ev_winc;            /* wait increment value */
   short _ev_sinc;            /* signal increment value */
   unsigned short _ev_link;   /* event use count */
   struct _evstr *queN;       /* next event in queue */
   struct _evstr *queP;       /* previous event in queue */
} event;

extern int
```

_ev_create(), _ev_link(), _ev_signal(), _ev_pulse(),
_ev_set(), _ev_setr(), _ev_unlink(), _ev_delete(),
_ev_info(), _ev_wait(), _ev_waitr(), _ev_read();

11) mode.h

```
/* modes.h        : file modes and permissions expressed in 8 bits */

#define S_IFMT    0xff    /* mask for type of file */
#define S_IFDIR   0x80    /* directory */
#define S_ISIZE   0x20    /* set initial file size */

/* permissions */
#define  S_IPRM    0xff     /* mask for permission bits */
#define  S_IREAD   0x01     /* owner read */
#define  S_IWRITE  0x02     /* owner write */
#define  S_IEXEC   0x04     /* owner execute */
#define  S_IOREAD  0x08     /* public read */
#define  S_IOWRITE 0x10     /* public write */
#define  S_IOEXEC  0x20     /* public execute */
#define  S_ISHARE  0x40     /* sharable */
```

12) module.h (extrait)

```
/* Module.h       : OS-9/68k module header definitions   */

#define VHPCNT  (sizeof(struct modhcom)-2)    /* sizeof common header */
#define MODSYNC 0x4afc                        /* module header sync code */
#define CRCCON  0x800fe3                      /* crc polynomial constant */

/* Module access permission values */
#define MP_OWNER_READ   0x0001
#define MP_OWNER_WRITE  0x0002
#define MP_OWNER_EXEC   0x0004
#define MP_GROUP_READ   0x0010
#define MP_GROUP_WRITE  0x0020
#define MP_GROUP_EXEC   0x0040
#define MP_WORLD_READ   0x0100
#define MP_WORLD_WRITE  0x0200
#define MP_WORLD_EXEC   0x0400
#define MP_OWNER_MASK   0x000f
#define MP_GROUP_MASK   0x00f0
#define MP_WORLD_MASK   0x0f00
#define MP_SYSTM_MASK   0xf000

/* Module Type/Language values */
#define MT_ANY      0
#define MT_PROGRAM  1
#define MT_SUBROUT  2
#define MT_MULTI    3
#define MT_DATA     4
#define MT_CSDDATA  5
#define MT_TRAPLIB  11
#define MT_SYSTEM   12
#define MT_FILEMAN  13
#define MT_DEVDRVR  14
#define MT_DEVDESC  15

#define ML_ANY      0
#define ML_OBJECT   1
#define ML_ICODE    2

#define mktypelang(type,lang)   (((type)<<8)|(lang))

/* Module Attribute values */
#define MA_REENT    0x80
```

```
#define MA_GHOST    0x40
#define MA_SUPER    0x20

#define mkattrevs(attr, revs)   (((attr)<<8)|(revs))

/* configuration module:  version smoothing definitions (_mcompat field) */
#define CP_SLOWIRQ     (1<<0)       /* save all regs during IRQ processing */
#define CP_NOSTOP      (1<<1)       /* don't use STOP instruction */
#define CP_NOGHOST     (1<<2)       /* don't retain ghost/sticky modules */
#define CP_NOBURST     (1<<3)       /* don't enable 68030 burst-fill */
#define CP_ZAPMEM      (1<<4)       /* patternize memory when allocated & freed */
#define CP_NOCLOCK     (1<<5)       /* don't start system clock during coldstart */

typedef unsigned short ushort;

struct modhcom {
    short   _msync,             /* sync bytes ($4afc) */
            _msysrev;           /* system revision check value */
    long    _msize,             /* module size */
            _mowner,            /* owner id */
            _mname;             /* offset to module name */
    short   _maccess,           /* access permission */
            _mtylan,            /* type/lang */
            _mattrev,           /* rev/attr */
            _medit;             /* edition */
    long    _musage,            /* comment string offset */
            _msymbol;           /* symbol table offset */
    short   _mident;            /* ident code for ident program */
    char    _mspare[12];        /* reserved bytes */
    short   _mparity;           /* header parity */
};

/* Executable memory module – or data module- */
typedef struct {
    struct modhcom _mh;         /* common header info */
    long    _mexec,             /* offset to execution -or data- entry point */
            _mexcpt,            /* offset to exception entry point */
            _mdata,             /* data storage requirement */
            _mstack,            /* stack size */
            _midata,            /* offset to initialized data */
            _midref;            /* offset to data reference lists */
} mod_exec;
```

13) stdio.h (extrait)

```
// “”””””””””””””””””””””””””””””””””””””””””””””””
#define _READ           1
#define _WRITE          2
#define _UNBUF          4
#define _BIGBUF         8
#define _EOF            0x10
#define _ERR            0x20
#define _SCF            0x40
#define _RBF            0x80
#define _DEVMASK        0xc0
#define _WRITTEN        0x0100    /*  buffer written in update mode  */
#define _MYBUF          0x0200    /*  system allocated buffer  */
#define _INIT           0x8000    /*  _iob initialized  */

#define EOF (-1)
#define EOL 13
#define NULL 0
```