MELISSA

TECHNICAL NOTE 18.3

C. BINOIS - 1993

REGISTRE DES ÉDITIONS
(DOCUMENT CHANGE LOG)

ÉDITION *(ISSUE)*	**Date**	**Observations**
00	05/93	ORIGINAL EDITION

RÉPERTOIRE DES PAGES
(PAGE ISSUE RECORD)

Ce document comprend les pages suivantes à l'édition indiquée
(Issue of this document comprises the following pages at the issue shown)

Document	**Pages**	**ÉDITION** *(ISSUE)*
TECHNICAL NOTE 18.3	ALL	00

TABLE DES MATIERES
(TABLE OF CONTENTS)

PAGE

I. BUT DU DOCUMENT

Ce document décrit le programme de contrôle dévelloppé pour le contrôle du compartiment photosynthétique de l'écosystème MELISSA. L'ensemble des problèmes rencontrés y est détaillé avec les solutions choisies. Les appellations relatives aux variables sont issues du manuel de programmation INDUSTAR. Le langage utilisé est le langage C. Le développement a été réalisé avec l'outil MICROSOFT C5.1, et l'ouvrage de B. KERNIGHAN reste la référence pour ce qui est des concepts de base du langage.

II. DÉFINITION DU PROGRAMME DE CONTRÔLE

Le contrôle d'un écosystème clos artificiel (tel que la boucle MELISSA dans le cadre d'une utilisation comme système de support vie) consiste à régler en temps réel les valeurs d'un groupe de paramètres, afin de réaliser un objectif fixé (production de biomasse). Pour cela, il est nécessaire de disposer d'un moyen autonome (ordinateur) capable d'acquérir et de traiter des données, pour réaliser les actions nécessaires.

Cette possibilité existe sur le système de contrôle INDUSTAR grâce à la station "GPS" (Global Purpose Station) ou encore"station blanche". L'utilisateur peut y développer ses propres applications en langage C.

Lors de l'écriture du programme nous avons été confrontés à différentes difficultés inhérentes à l'utilisation des procédures GPS. Afin de rendre cet interface plus conviviale nous avons créé des procédures utilitaires génériques. Ainsi, le programme a été décomposé en trois modules (figure 2.1) :

- un module de gestion générale, chargé du cadencement des opérations. Il comprend une interface utilisateur, pour l'affichage d'informations, telles que les résultats du contrôle, et l'état du programme.

- un module d'accès et de gestion des données (régulateurs P100), qui simplifie l'utilisation des procédures de la bibliothèque "GPS".

- un module de contrôle réalisant le contrôle du compartiment photosynthétique, qui constitue la partie active du programme.

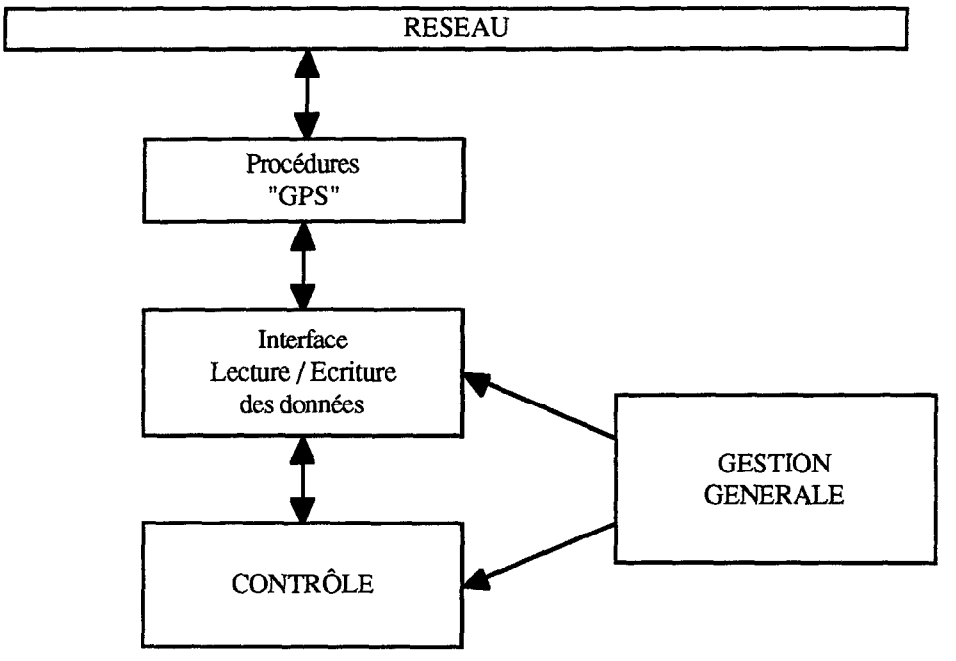

Figure 2.1 - Architecture du programme de contrôle.

Le programme décrit dans cette note est écrit en langage C et utilise l'environnement existant du système INDUSTAR. Dans les page qui suivent, les concepts et les notations utilisées sont celles du langage C Microsoft version 5.0.

II.1. GESTION GÉNÉRALE

Cette partie du programme est chargée de fournir un environnement permettant a l'utilisateur de développer le contrôle du (ou des) procédé sans se préocuper de tâches annexes. Le programme doit réaliser principalement le contrôle d'un (ou plusieurs) procédés, cette partie gére donc l'enchainement des taches à effectuer dont la description figure ci-après.

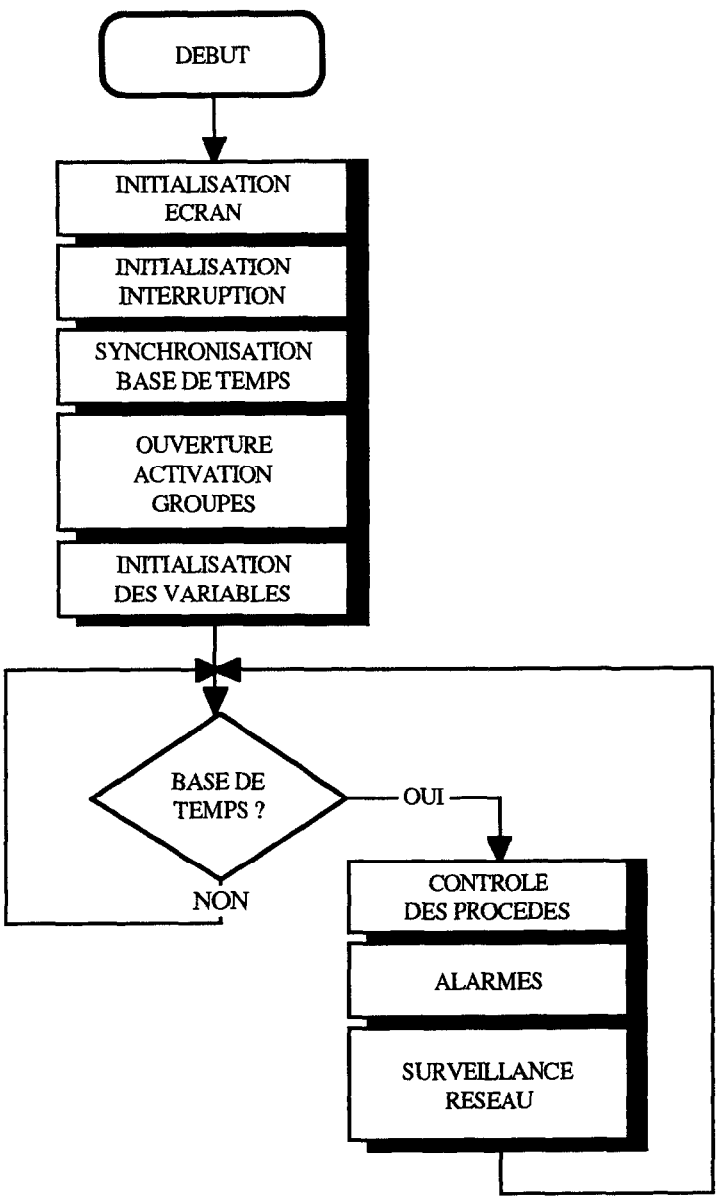

Figure 2.1.1./1 - Organigramme général du programme de contrôle.

II.1.1. INITIALISATION ET GESTION DE L'ÉCRAN

Pendant l'exécution du programme, ainsi que durant la phase de mise au point, il est utile de pouvoir visualiser un certain nombre d'informations à l'écran telles que :

- le groupe activé,
- la date et l'heure,
- l'état du programme et la procédure en cours d'exécution,
- les résultats du contrôle, la valeur de certaine variables,
- les messages d'erreur et les messages d'ordre général.

Nous avons donc réalisé une interface homme <-> machine. L'écran est découpé en plusieurs zones dont une (la plus grande) est réservée a l'affichage des résultats relatifs au controle du procédé.

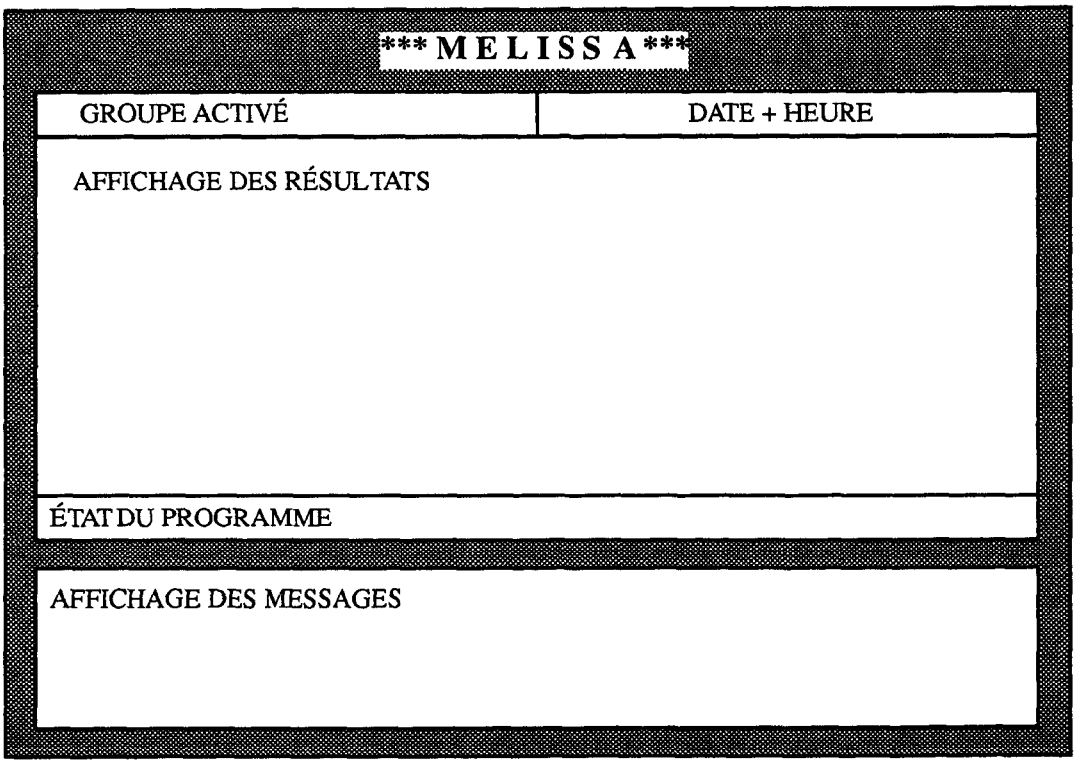

Figure 2.1.1./1 - Découpage de l'écran d'affichage.

Le curseur est positionne par défaut dans la fenêtre messages, ce qui permet d'afficher à tout moment une chaîne de caractères avec la commande "_outtext(*chaîne*);" sans modifier l'affichage général.

Procédures contenues dans le fichier screen.c

Procédure	Fonction
- screen_init()	Initialisation de l'écran, dessin des cadres, définition des différentes fenêtres d'affichage.
- display_activ_group(GPS_FILE *)	Affiche le groupe active
- display_no_activ_group()	Efface le nom du groupe affiche
- display_time(char *)	affiche la date et l'heure, la chaîne de caractères est formatée par la fonction time_base()
- display_result(char *,short x,short y)	affiche la chaîne de caractère a partir de la coordonnée (x,y) dans la fenêtre d'affichage des résultats.
-display_status(char *)	affiche la chaîne de caractère sur la ligne d'affichage des status.
- display_error(char *)	affiche la chaîne de caractères en rouge clignotant sur fond vert dans la fenêtre ou se trouve le curseur, une alarme sonore est également émise.
- use_message_window()	positionne le curseur dans la fenêtre d'affichage des messages.
- use_group_window()	positionne le curseur dans la fenêtre d'affichage du groupe.
- use_time_window()	positionne le curseur dans la fenêtre d'affichage de la date.
- use_status_window()	positionne le curseur dans la fenêtre d'affichage de l'état du programme.
- use_display_window()	positionne le curseur dans la fenêtre d'affichage des résultats
- tab(short x,short y)	positionne le curseur a la coordonnée (x,y) dans la fenêtre ou il se trouve.

II.1.2. INTERRUPTION du PROGRAMME

L'interruption d'un programme en langage C n'est pas implicite avec la commande "CONTROL C". Il faut définir le vecteur d'interruption, sur la procédure appelée en cas d'utilisation de cette commande. La procédure d'interruption a pour but de définir les instructions à réaliser lorsque l'utilisateur désire stopper l'exécution du programme, et de lui donner la possibilité de stopper ce programme.

Dans un premier temps cette procédure permet l'arrêt (après confirmation) du programme. Il est possible de l'étoffer en donnant la possibilité de modifier un ou plusieurs paramètres de contrôle lors de l'appel de cette procédure. Cependant, le nombre de paramètre devant êtres réglés par l'utilisateur est souvent faible, il est d'ailleurs possible d'utiliser les zones mémoires disponibles du MICON (LOC accessibles sur une station contrôle / commande). Ceci présente l'avantage de pouvoir bénéficier de l'archivage des données au sein du système INDUSTAR sans interrompre le déroulement du programme, et, est beaucoup plus convivial.

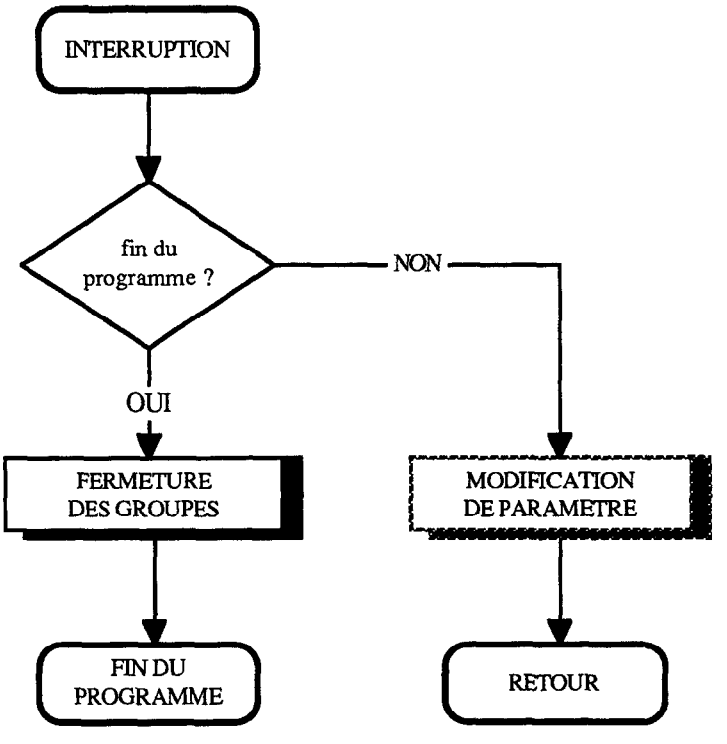

Figure 2.1.2./1 - Synoptique de la procédure d'interruption.

II.1.3. BASE DE TEMPS

Le rôle de la base de temps est d'intégrer la variable temps au sein du programme. Il s'agit en particulier de fournir des tops au programme de contrôle, afin de lui faire enchaîner les opérations de contrôles suivantes :

- l'acquisition des variables.

- la détermination des nouvelles consignes (intensité de l'éclairage,...).

- l'envoi des nouvelles consignes vers les régulateurs.

Nous avons choisi une base de temps de 1 min, car le processus évolue lentement, et ceci correspond à la période d'archivage de la plupart des données. Toutefois cette valeur reste modifiable dans le fichier "melissa.h". Cette procédure utilise l'horloge interne du PC grâce aux fonctions C relatives au temps. Le premier appel de cette procédure entraîne une synchronisation de la base de temps à la minute ronde. Cette étape permet à l'utilisateur de connaître précisément le top suivant (15:05:00, par exemple, au lieu de 15:05:12), et facilite le suivi du programme.

Procédures contenues dans le fichier melfct.c

- timebase()	Base de temps, retourne 0 lorsque l'échantillonnage des données doit être effectué, sinon retourne le temps de l'horloge interne en secondes. Cette fonction assure le formatage de la chaîne permettant d'afficher la date et l'heure et appelle la fonction display_time().
- wait_time(int i)	Génère une pause de i secondes

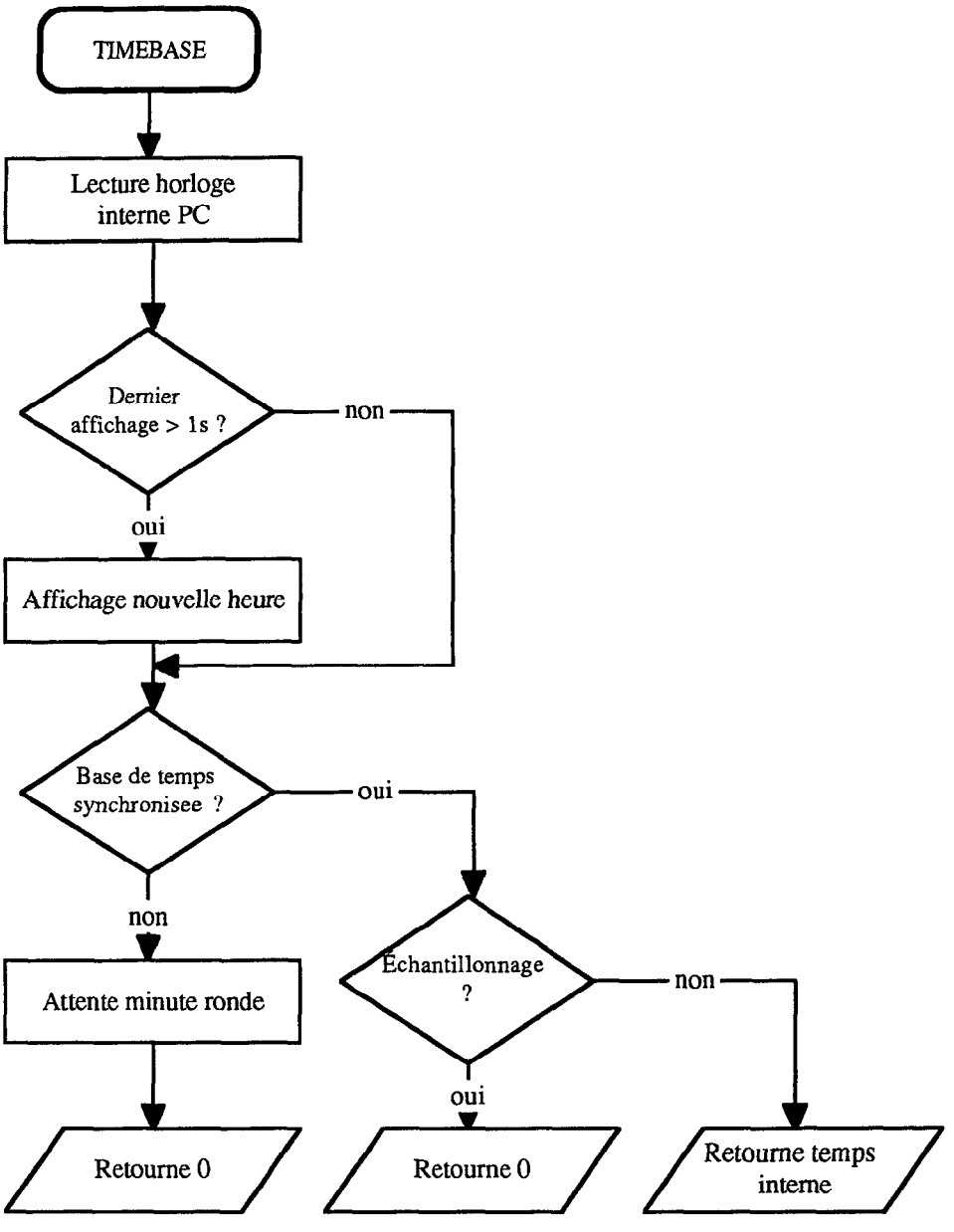

Figure 2.1.3./1 - Synoptique de la fonction base de temps.

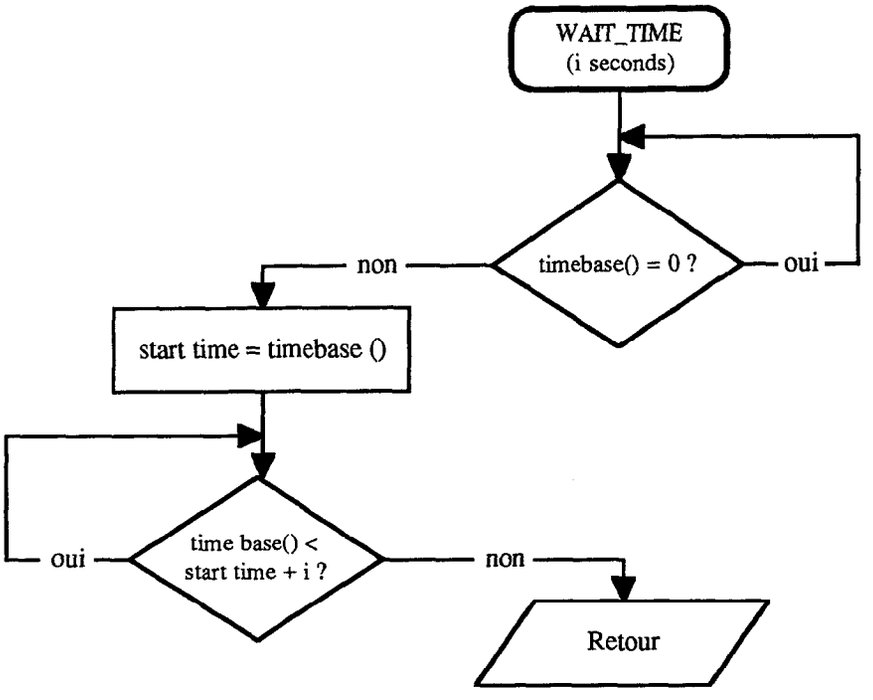

Figure 2.1.3./2 - Synoptique de la fonction wait_time.

II.1.4. INITIALISATION DES VARIABLES

Chaque variable utilisée est associée à une structure de type VARS (voir §III). Il est nécessaire de renseigner un certain nombre de champ de cette structure avant de pouvoir l'utiliser. L'initialisation commence par écrire le champ "name" qui est le seul défini par l'utilisateur. Ensuite une acquisition de toutes les variables déclarées met à jour automatiquement tous les autres champs (si nécessaire).Pour les variable dont on n'utilise que la valeur instantanée, cette procédure suffit. Mais pour celles dont on a besoin des valeurs passées le problème est que le tableau val[] est a priori initialise a zéro au moment du lancement du programme et ne contient donc pas les valeurs passées. Il a deux possibilités de gérer ce problème :

-Attendre que l'on ait acquis suffisamment de points pour commencer le contrôle(au moins une heure pour la mesure de vitesse par exemple).

-Initialiser le tableau val[] de façon a pouvoir commencer le contrôle dès le lancement du programme. L'hypothèse faite pour remplir ce tableau est de supposer que le contrôle était effectue de façon parfaite avant le lancement du programme. Ceci signifie que les variables ainsi que les pentes des compteur étaient stables auparavant. Les tableaux val[] seront initialises de la façon suivante :

- avec la valeur courante pour les variables

- à partir de la valeur courante en tenant compte de la pente calculée a l'aide du modèle pour les compteurs.

Cette procédure d'initialisation permet de commencer le contrôle du réacteur dès le lancement du programme.

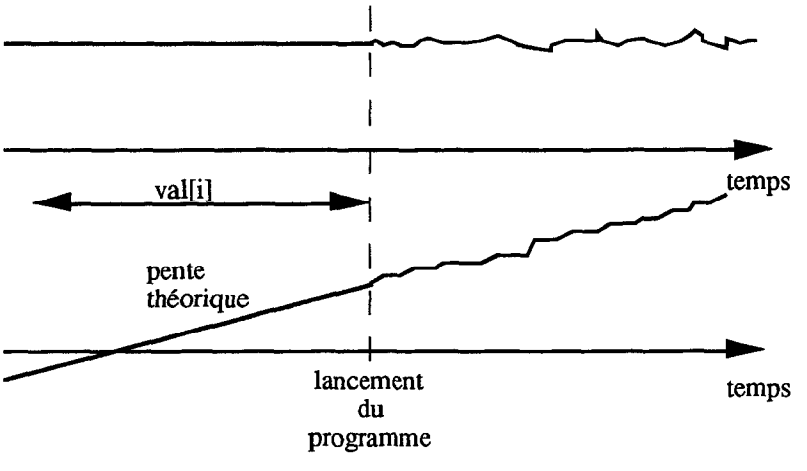

Figure 2.1.4./1 - Principe d'initialisation des variables (en haut) et des compteurs (en bas).

II.1.5. GESTION des ALARMES

La boucle MELISSA est conçue pour fonctionner en permanence durant des périodes de plusieurs milliers d'heures. Il y a donc une probabilité élevée pour qu'il y ait des défaillances. Le système de contrôle INDUSTAR possède une gestion centralisée des alarmes issues des contrôleurs. Cependant ces alarmes ne prennent en compte que les variables vues par les contrôleurs et les paramètres définis par l'utilisateur. Tous les cas de panne possibles ne sont pas détectés par les contrôleurs (limitation de la visibilité des variables et du nombre de pas de programme), de la même façon une alarme au niveau 0 peut être sans effet sur les niveaux supérieurs.

La gestion des alarmes au niveau du programme de contrôle a donc pour objectifs :

- Analyser les alarmes issues des contrôleurs afin de stopper éventuellement le programme de contrôle et de les prendre en compte.

- Détecter les pannes complexes Prendre des dispositions pour informer et assurer la sauvegarde des cultures (en modifiant les consignes par exemple). Générer une alarme pour informer l'utilisateur, permettant ainsi son intervention dès que possible A chaque nouveau problème rencontré, celui-ci devra être décrit dans le programme pour permettre une détection ultérieure. Cette partie du programme est donc étroitement liée à l'architecture matérielle de la culture

Les variables de base utilisées pour le traitement des alarmes sont données dans le tableau I.

REPÈRE	DESCRIPTION
LOC-0121	Vecteur alarme MICON 1
LOC-0122	Status alarme MICON 1
VD--0142	RAZ alarme P100 N°1
VD--0145	RAZ alarme P100 N°2
VD--0149	RAZ alarme P100 N°3
VD--0153	RAZ alarme P100 N°4
VD--0142	RAZ alarme MICON 1
VD--0143	RAZ alarme générale
VD--0146	Activation lampe

Tableau 2.1.5./1 - Variables relatives aux alarmes du MICON 1.

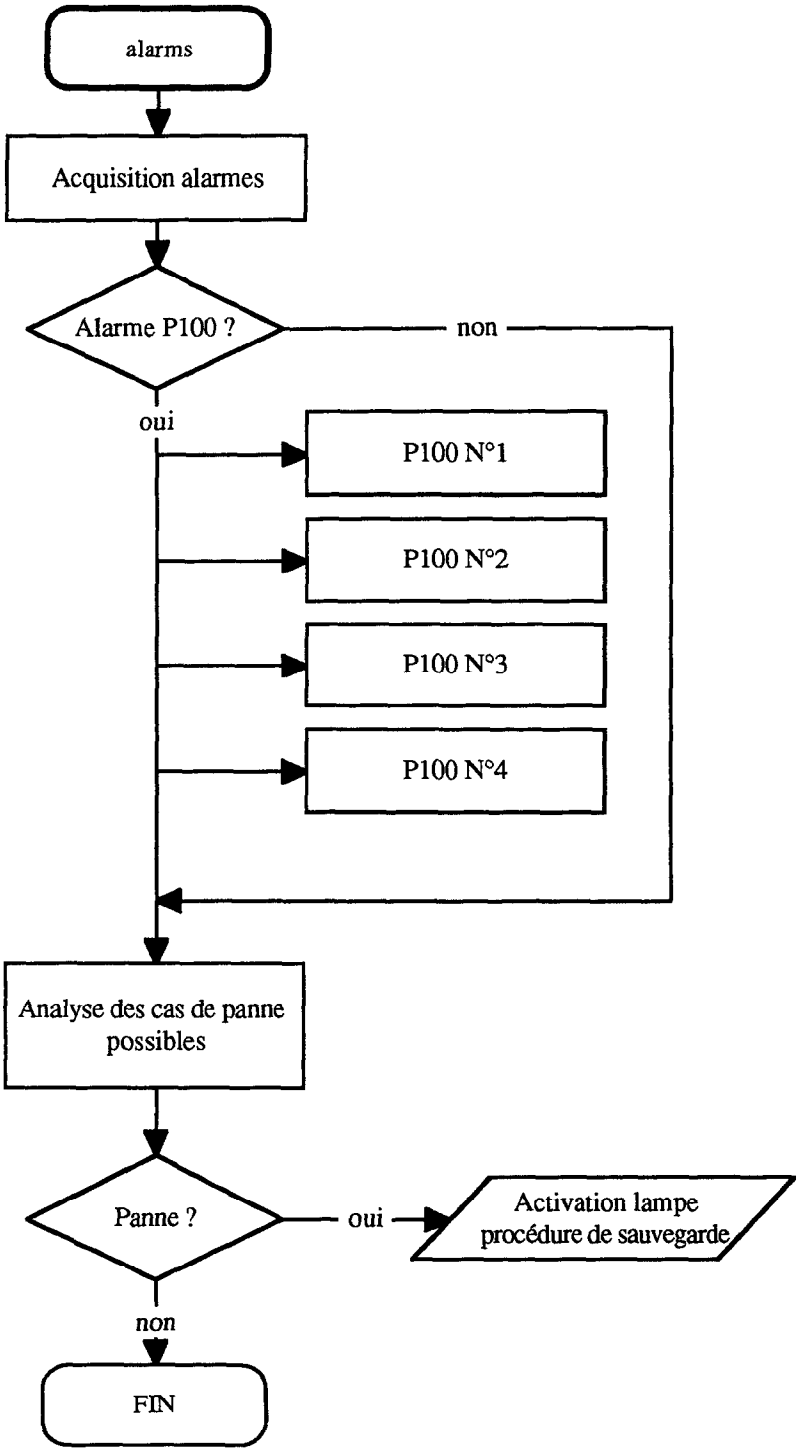

Figure 2.1.5./2 - Organigramme de la procédure de traitement des alarmes.

La compilation des fichiers sources constituant le programme de contrôle avec le fichier alarm.c produit un fichier exécutable trop volumineux pour fonctionner avec l'espace mémoire disponible (≈ 2MO). Le fichier alarms.c a été testé dans une configuration allégée du programme puis a été supprimé pour permettre la fin du développement. Ce fichier devra être réintégré lorsque la capacité mémoire le permettra.

II.1.6. SURVEILLANCE DU RÉSEAU, GESTION DES ERREURS

La surveillance du réseau se limite a vérifier que les données sont bien rafraîchies dans la boite aux lettres à l'aide de la fonction garde. L'absence d'informations détaillées sur la gestion du réseau ne permet pas d'envisager des opérations plus complètes. De même la descriptions des codes erreur retournés par les procédures manquent de clarté. Dans la plupart des cas, en cas d'erreur, celle-ci est soit ignorée (mais indiquée à l'utilisateur dans la fenêtre message) soit entraîne un arrêt du programme.

II.2. GESTION des DONNÉES

L'interface de lecture et d'écriture des données est l'élément essentiel qui permettra d'assurer la souplesse du développement du programme de contrôle. Il doit être fiable, c'est à dire capable de détecter les erreur liées à l'accès aux données, aussi bien que les erreurs de l'utilisateur. D'autre part cet interface doit simplifier l'accès aux données et permettre une évolution aisée du programme de contrôle.

II.2.1. INTERFACE DE GESTION GPS

Les données disponibles sur les P100 sont de deux types : les "Repères" accessibles en lecture seulement, et les "Commandes" accessibles en lecture et en écriture. Les données sont rassemblées par bloc de 40 maximum dans des fichiers appelés "fichiers groupe". Ceux-ci, sont constitués sur la station système, et contiennent le nom des données, et les informations utilisées par les procédures "GPS".

Ces procédures sont :

open_gr permet l'ouverture d'un fichier groupe. Il est possible d'ouvrir plusieurs groupes à la fois.

activ_gr active le groupe spécifié, celui-ci doit être ouvert au préalable. Un seul groupe peut être activé à la fois, autorisant ainsi la lecture et / ou l'écriture des variables (40 maxi.) qu'il contient.

rd_gps lit le "Repère" spécifié, et met à jour *l'union* "IGPS".

set_cmd lit la "Commande" spécifiée, et met à jour *l'union* "OGPS".

wr_gps écrit la nouvelle valeur de la "Commande", à l'aide des *unions* "OGPS" et "S_USER" .

close_gr ferme le fichier groupe spécifié.

gettags renvoie le nom de la variable de rang "i" dans le groupe activé. Il est ainsi possible de vérifier la présence d'une donnée dans ce groupe.

garde vérifie la mise à jour des données dans la boite aux lettres.

" Une *union* est une variable qui peut contenir (à différents moments) des objets de types et de tailles différents. Les *unions* fournissent un moyen de manipuler plusieurs types de donnnées dans une même zone mémoire, ce qui permet de réduire l'espace utilisé."

L'utilisation des procédures "GPS" présente deux inconvénients majeurs :

- Il faut être sûr que le groupe dans lequel la donnée est déclarée, est effectivement bien activé.

- Il faut connaître la nature de la donnée, pour déterminer la procédure d'accès et la *structure* mise à jour.

Ces contraintes imposent à l'utilisateur de lier durablement une variable avec la *structure* d'accès. Ceci pénalise l'évolution et la souplesse du programme de contrôle. Par exemple, un changement d'affectation dans les groupes, conduit à un nombre important de modifications dans les fichiers sources, d'où un risque d'erreurs. C'est pourquoi, nous avons développé un interface permettant :

- l'uniformisation de la représentation des données manipulées par le programme de contrôle (*structure* "VARS").

- la simplification de la gestion des groupes (ouverture et activation).

- la simplification des accès, à l'aide de deux procédures distinctes, utilisant la même *structure* pour la lecture, et pour l'écriture de tous les types de variables :

-read_var(* nom_variable)

-write_var(* nom_variable)

Cet interface utilise la *structure* "VARS" définie plus loin, et comprend deux parties :

- Gestion des groupes.

- Accès aux données.

II.2.2. GESTION DES GROUPES

Il est possible d'ouvrir plusieurs groupes (dans les limites tolérées par DOS) alors qu'un seul ne peut être activé à la fois (donnant accès à un maximum de 40 repères). Cette fonction est chargée de l'ouverture de tous les groupes définis par l'utilisateur et de l'activation de l'un d'eux. A l'ouverture d'un fichier *.GPS, la procédure retourne un identificateur DOS qu'il faut mémoriser pour pouvoir activer le groupe correspondant ou bien le clore. On envisage d'ouvrir plusieurs groupes en même temps. Il faut donc sauvegarder autant d'identificateurs que de fichiers ouverts pour pouvoir effectuer une recherche.

Une solution plus souple d'emploi est d'associer a chaque fichier ouvert une structure, et de chaîner ces structures entre elles de façon circulaire (la dernière pointe sur la première). Il ne reste alors plus qu'a mémoriser et manipuler qu'un seul pointeur (sur la structure correspondant au groupe active) pour effectuer les recherches.

la structure GPS_FILE est définie de la façon suivante :

- char	file	nom du fichier groupe *.GPS
- int	handler	identificateur DOS du fichier groupe associe
- int	rank	rang d'ouverture du fichier utilise lors de la recherche d'un repère
- struct _gps_file *next		pointeur sur la structure correspondant au fichier suivant

Spécification des fichiers groupe a ouvrir

Afin de ne pas alourdir le programme et de garder une certaine souplesse dans son utilisation, l'utilisateur a la possibilité de spécifier le nom des fichiers groupe a ouvrir hors du programme. Ceux-ci doivent avoir comme nom la syntaxe suivante:

[NOM]DD.GPS

DD = 01, 02, 03 etc. par ordre croissant a partir de 01

exemple SPIRU01.GPS SPIRU02.GPS SPIRU04.GPS
01 et 02 sont ouverts 04 ne l'est pas.

Ceci permet d'avoir un nom par compartiment et pour chacun plusieurs fichiers groupes (contrôle 01 et alarme 02 par exemple)

La spécification se fait dans le fichier GPS.FIL en mettant la ligne suivante autant de fois que nécessaire

BASE:[NOM] exemple BASE:SPIRU

ERREUR DÉTECTÉES

- erreur de syntaxe dans GPS.FIL
- absence de GPS.FIL
- aucun fichier groupe n'a pu être ouvert

Procédures contenues dans le fichier GPSFILE.C

open_file_gps() Cette procédure ouvre tous les fichiers groupe déclarés dans GPS.FIL (si il existent) et réalise un chaînage circulaire à l'aide de la structure GPS_FILE (figure xx)

activ_grp_gps() Utilise le pointeur gps pour sélectionner la structure GPS_FILE suivante et active le groupe correspondant, retourne la nouvelle valeur du pointeur gps

close_grp_gps() ferme tous les groupes ouverts avec la fonction open_file_gps()

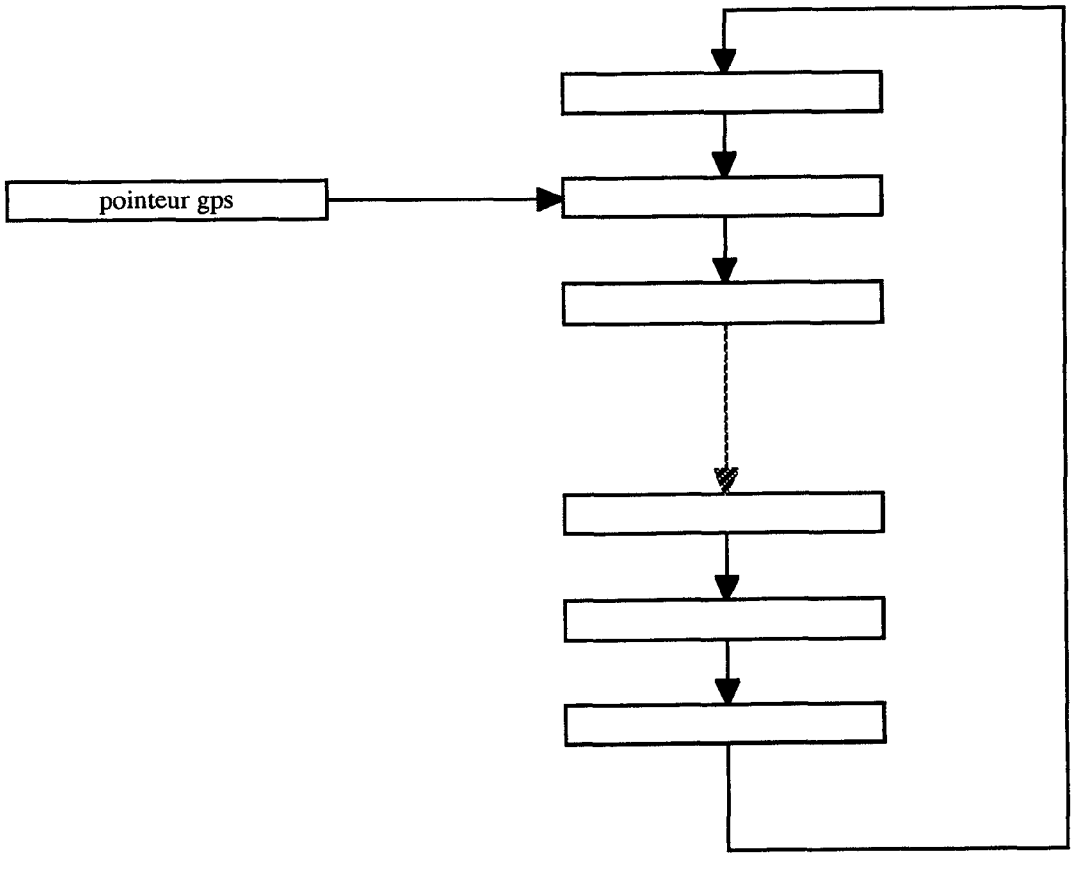

figure 2.2.2./1 - chaînage des structures GPS_FILE

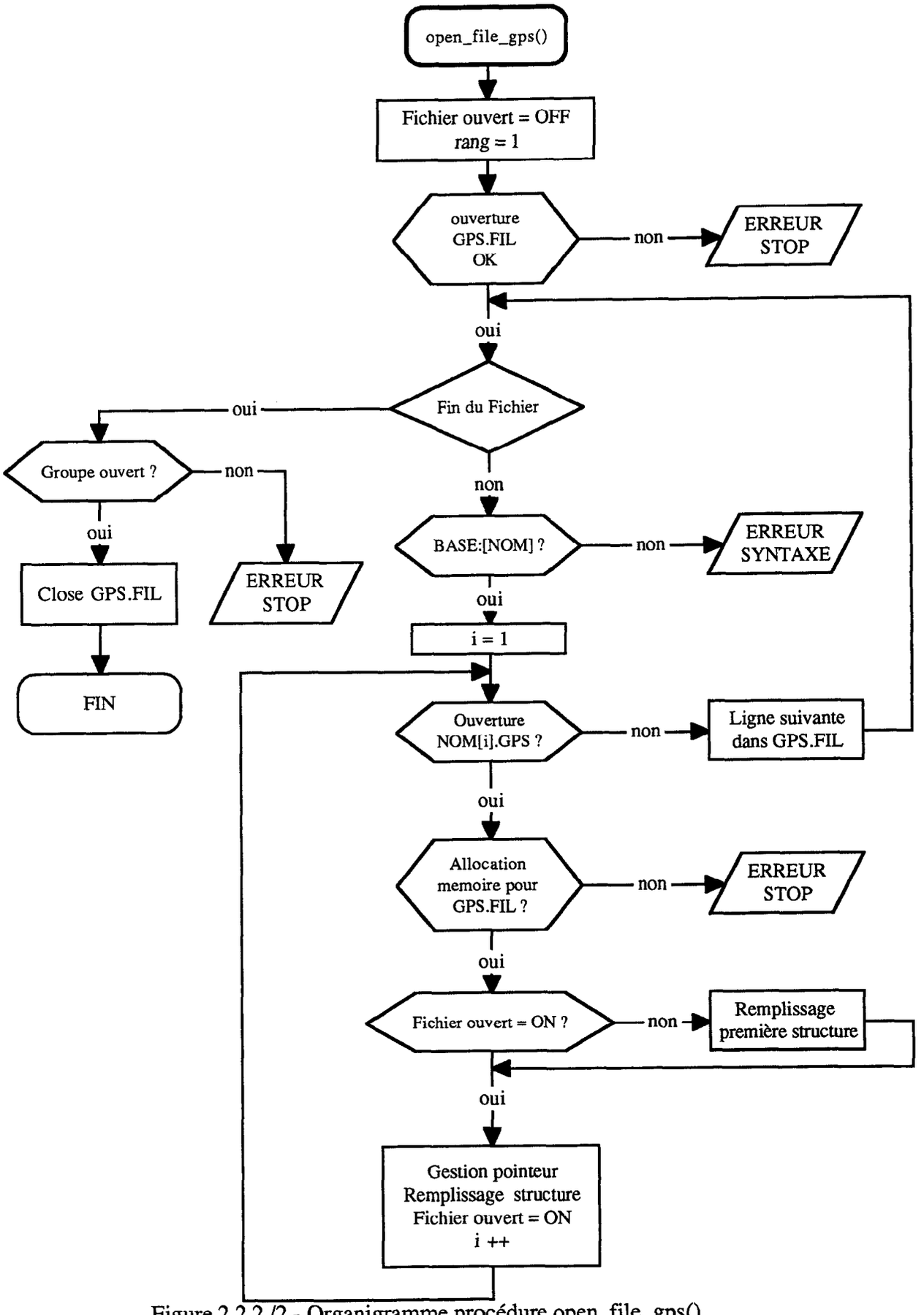

Figure 2.2.2./2 - Organigramme procédure open_file_gps().

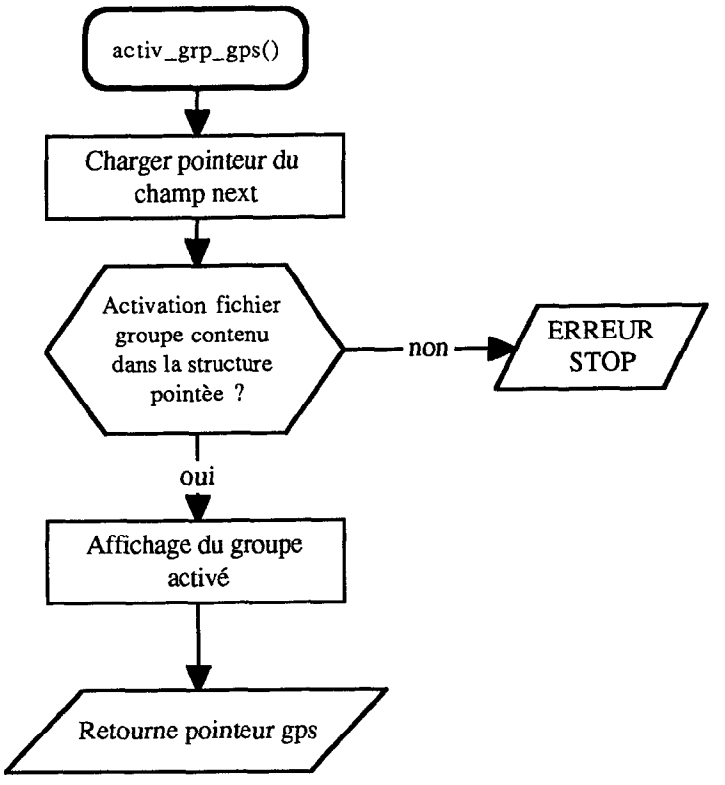

Figure 2.2.2./3 - Organigramme de la procédure activ_grp_gps().

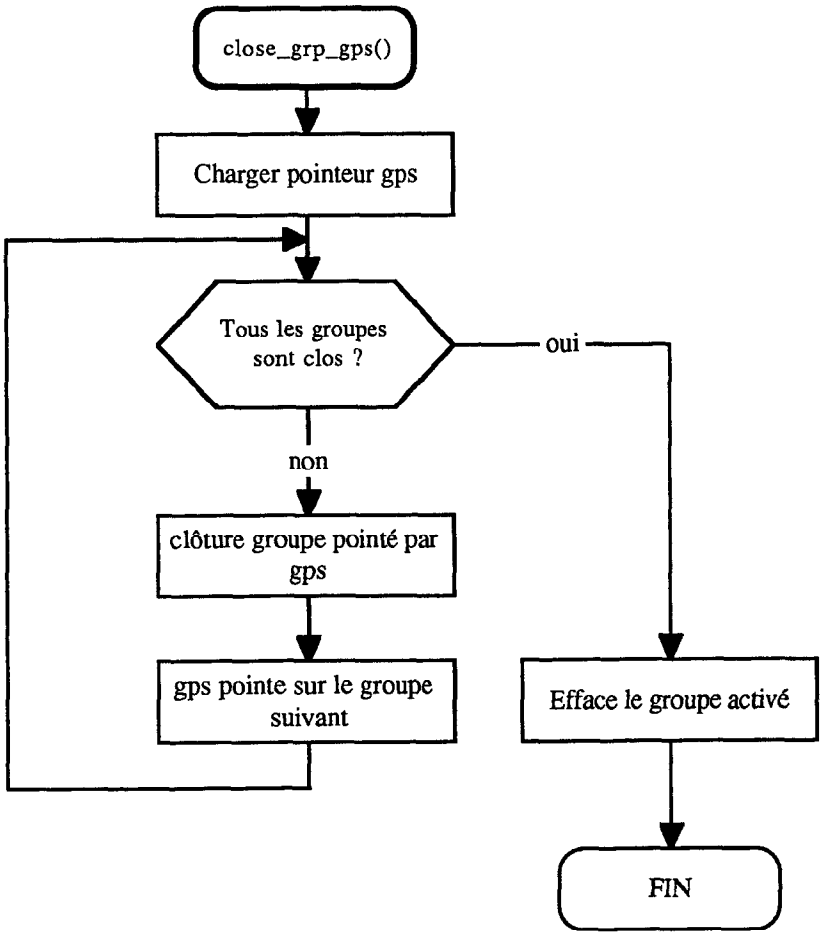

Figure 2.2.2./4 - Organigramme de le procédure close_grp_gps.

II.2.3. ACCÈS aux DONNÉES

II.2.3.a LA STRUCTURE "VARS"

Dans le but d'uniformiser la représentation des données manipulées par le programme, il a été nécessaire d'élaborer une structure utilisable pour toutes les variables MICON du système INDUSTAR, cette structure permet :

- de lire ou écrire une valeur par l'intermédiaire du repère déclaré dans le système de contrôle,

- d'utiliser les échantillons passés pour calculer une variation ou une pente compteur par exemple.

cette structure contient les champs suivants :

name	char	nom du repère ou de la commande, ce champ est le seul qui doit être spécifié par l'utilisateur à l'aide de l'instruction: sprintf(*variable*.name,"LOC-0125"); par exemple N.B. le nom du repère doit être spécifié en **MAJUSCULES** avec exactement 8 caractères

Les autres champs de la structure sont initialisés par la procédure de lecture lors du premier appel de cette procédure. Ces champs sont décrits ci-après :

nom	**type**	**description**
file	char	fichier groupe dans lequel le repère est déclaré.
type	int	code retourné par la procédure rd_gps ou set_cmd, une fois identifié il permet de déterminer la sous procédure a appeler pour effectuer la lecture ou l'écriture.
tag_cmd	unsigned char	ce champ permet de différencier les repères et les commandes
dev_num	unsigned int	donne le numéro de l'équipement gérant le repère, permet de tester si la boite aux lettres a été mise a jour.
value	double	valeur courante de la variable, mise à jour à chaque lecture.

i	int	le tableau val[] permet de stocker les valeurs passées de la variable. la gestion est assurée par le pointeur i (qui indique le dernier échantillon entré) sous forme d'un buffer circulaire. La gestion simplifiée de i implique que la dimension de val[] soit 2^n (n entier).
val[]	double	
min	double	valeurs minimum et maximum autorisées pour la variable si c'est une LOOP ou une LOC, si la variable est une VD alors max=activ_state.
max	double	
sp	double	ce champ est bidirectionnel, il sert à lire le set-point d'une LOOP ainsi qu'a écrire une valeur ou un set-point dans une commande. afin de ne pas perturber le système pour LOC et VD, sp contient la valeur courante. Si ce champ n'est pas modifié l'écriture de la variable est sans effet.
out	double	donne le pourcentage de l'action de sortie pour les boucles uniquement (variable de type LOOP).
unit[5]	char	unité des variables analogiques.
update	char	permet à la procédure de lecture de déterminer si la totalité de la structure a été mise à jour (update = ON), donc, si la structure est utilisable.

II.2.3.b PROCÉDURES D'ACCÈS

Les deux procédures (lecture et écriture) sont structurées de la même façon. Elles se décomposent en une procédure de gestion et une procédure d'accès.

La procédure de gestion a pour rôle d'appeler la procédure d'accès, et en cas d'erreur gère l'activation d'un nouveau groupe, pour tenter d'accéder la variable. La recherche de la variable dans les fichiers groupe est limitée au nombre de groupes ouverts (champ rank de la structure GPS_FILE).

La seule contrainte imposée à l'utilisateur est de définir le nom de la variable dans la procédure d'initialisation ; sprintf(*variable*.name,"LOC-0125"); de façon à lier les variables programme avec les variables système.

Procédures contenues dans le fichier VARS.C :

procédure	**description**
set_gps(* GPS_FILE)	permet de récupérer le pointeur *gps dans le segment résident des procédures d'accès, *gps est une variable globale de ce segment
read_var(* VARS)	procédure de gestion pour la lecture d'une variable
read_gps(* VARS)	cette procédure est chargée de déterminer les instructions à exécuter pour lire la variable spécifiée. Lorsqu'il s'agit du premier accès, un phase d'identification du type de variable est nécessaire pour mettre à jour la structure VARS ; les variables globales nbtags et nbcommands sont utilisées pour différencier le type TAG du type CMD avec la fonction gettag. En cas d'échec on effectue un retour à la procédure read_var qui demande l'activation d'un autre groupe si nécessaire. Une fois le type identifié, on appelle la fonction de lecture correspondante (rd_gps ou set_cmd). Le code retourné est mémorisé dans VARS, et permet de savoir quelle est la structure à utiliser dans les unions IGPS et OGPS pour effectuer la mise a jour de la structure VARS.
write_var(* VARS)	procédure de gestion pour l'écriture d'une variable
write_gps(* VARS)	cette procédure est chargée de déterminer les instructions à exécuter pour écrire la variable spécifiée. Si la structure VARS n'a pas été initialisée elle appelle read_var pour le faire.

II.2.3.b.1 LECTURE

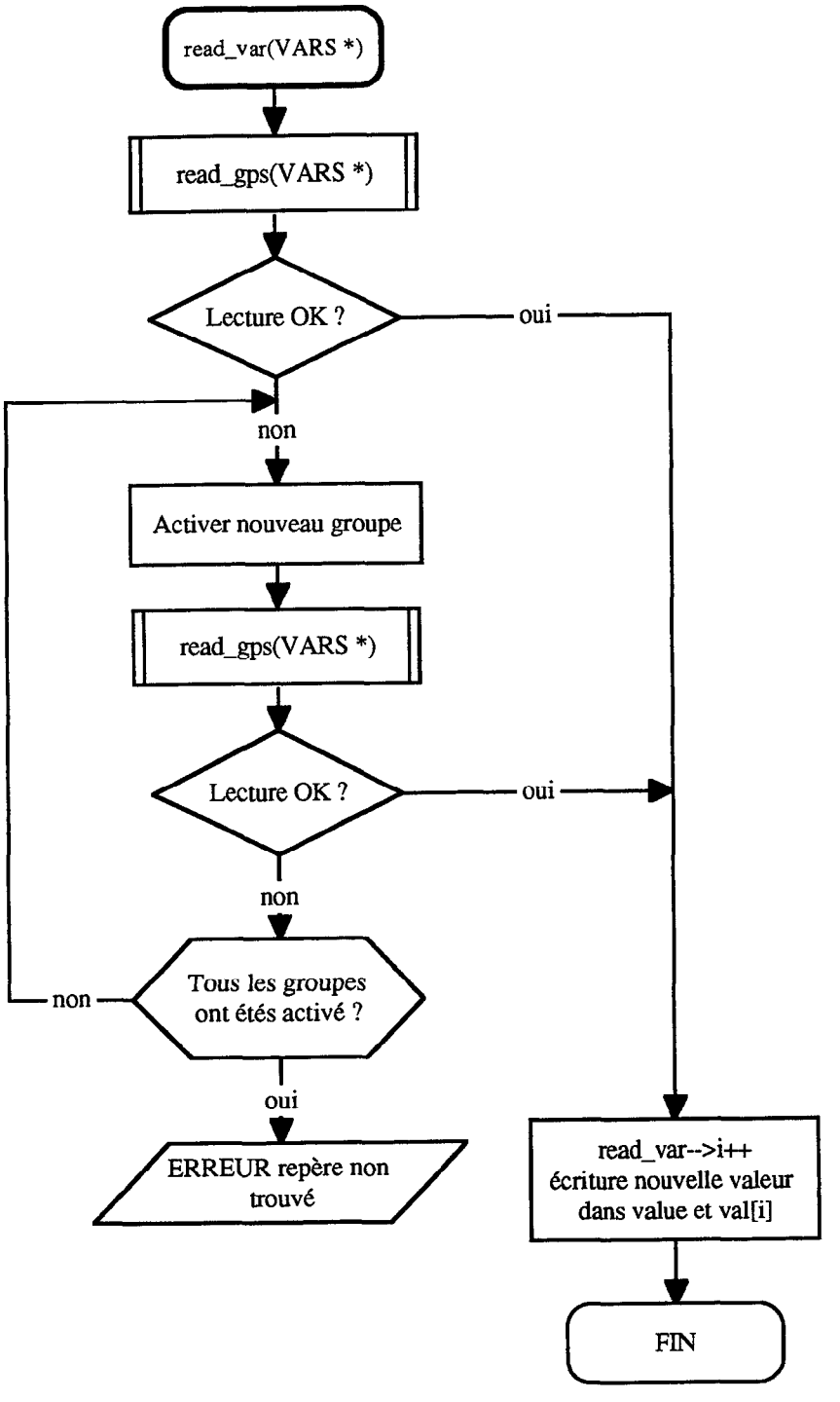

Figure 2.2.3./1 - Organigramme de la procédure read_var.

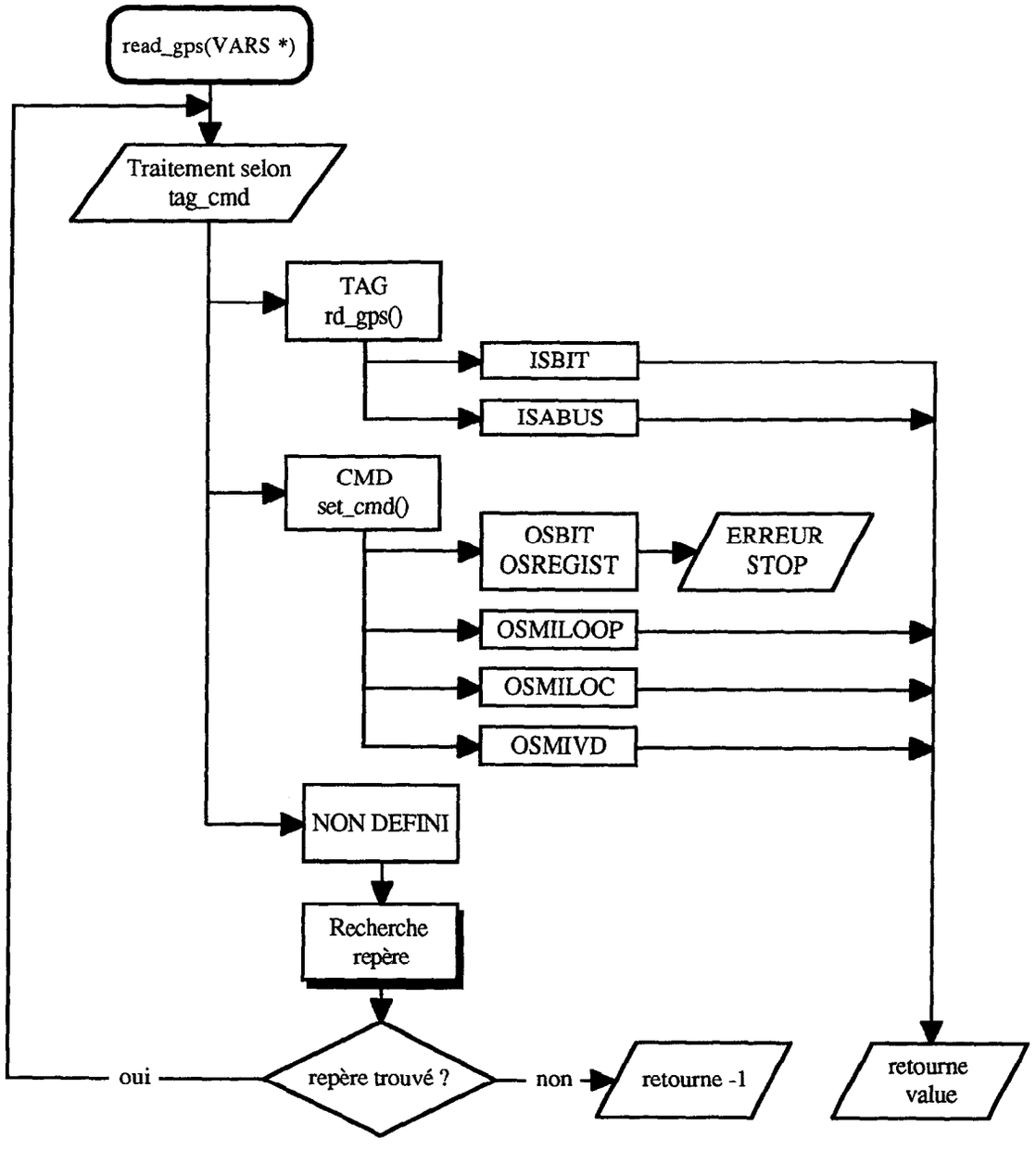

Figure 2.2.3./2 - Organigramme de la procédure read_gps()

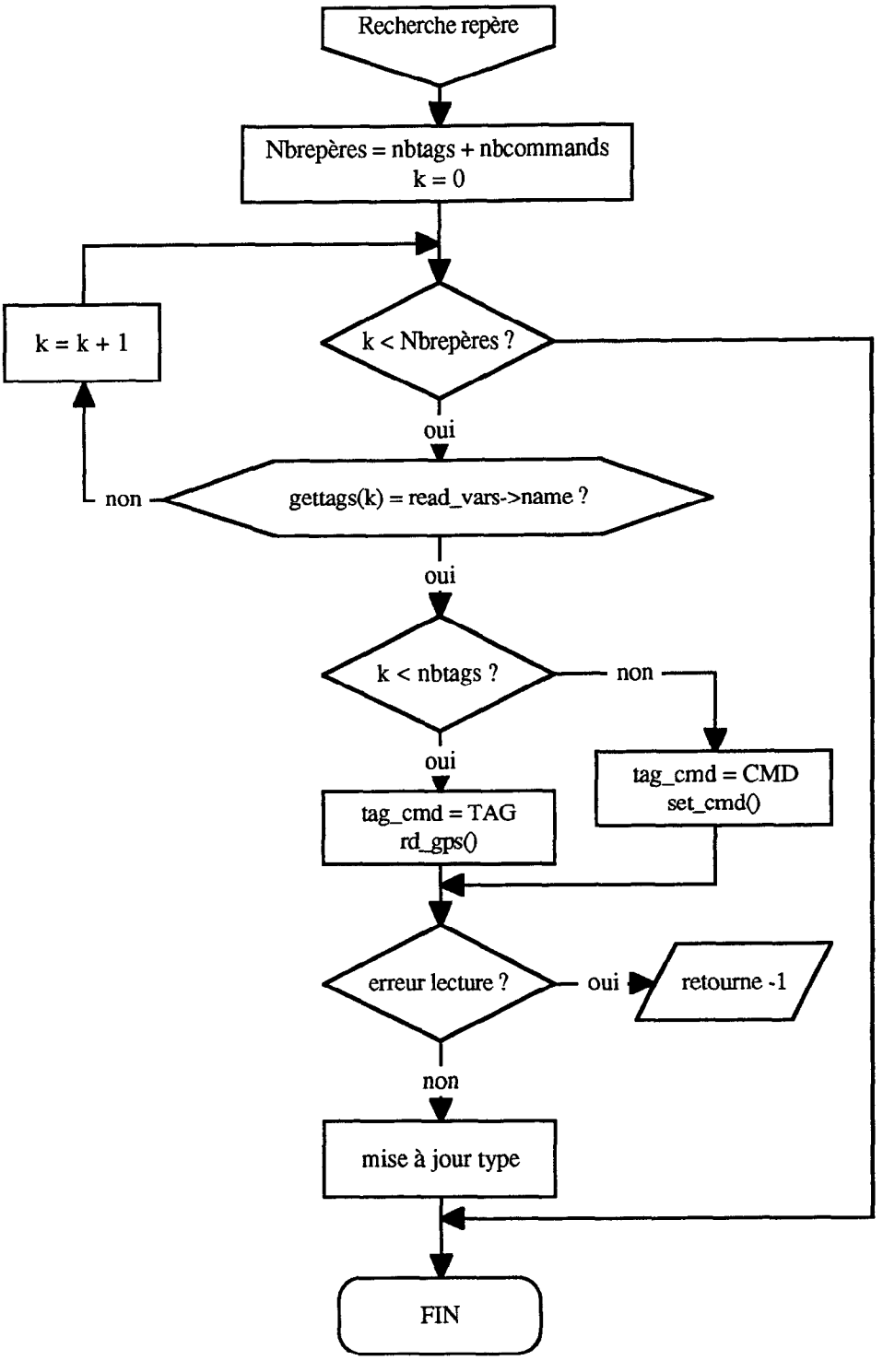

Figure 2.2.3./3 - Organigramme de la routine de recherche de repère

II.2.3.b.2 ÉCRITURE

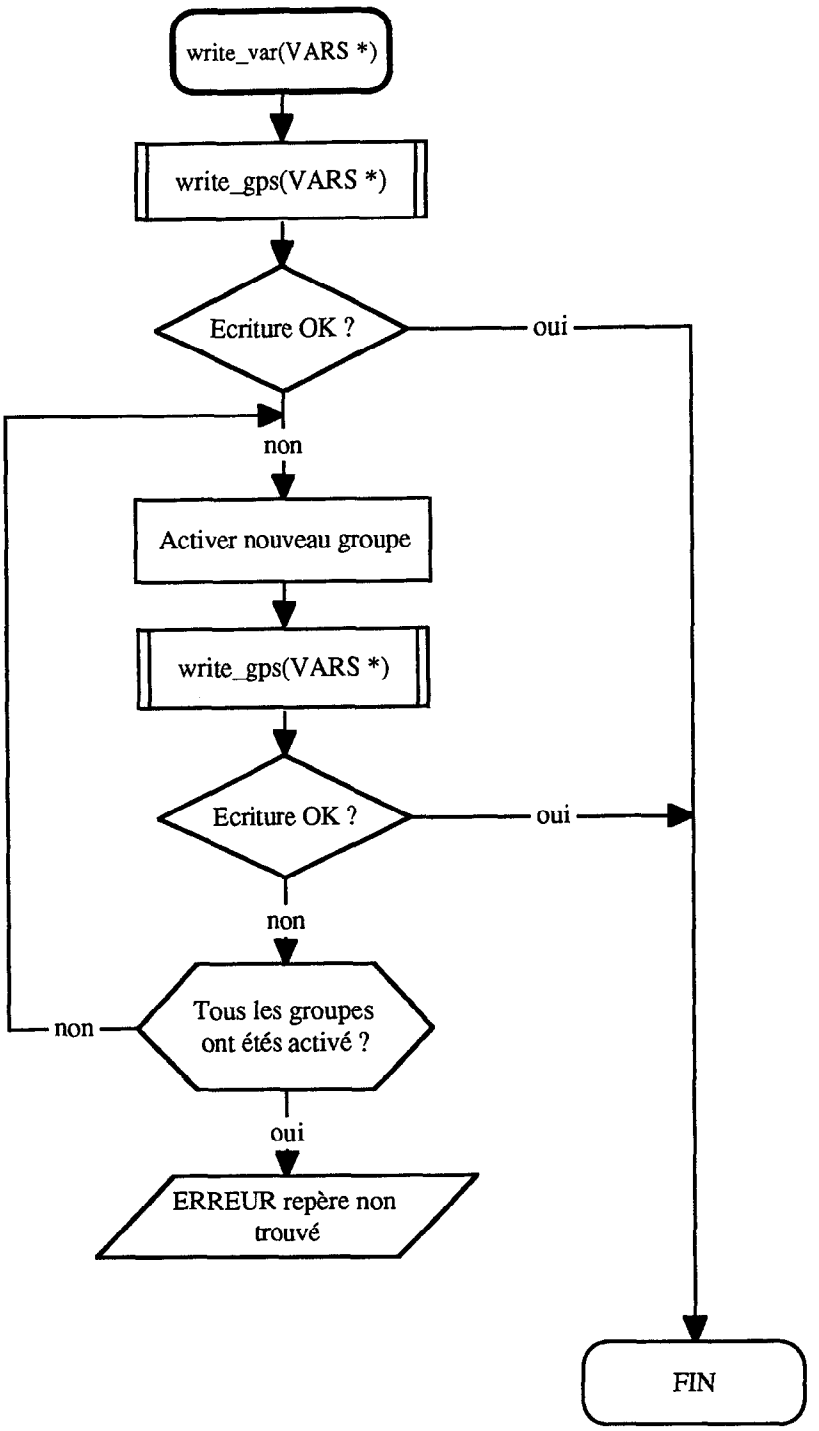

Figure 2.2.3./4 - Organigramme de la procédure write_var

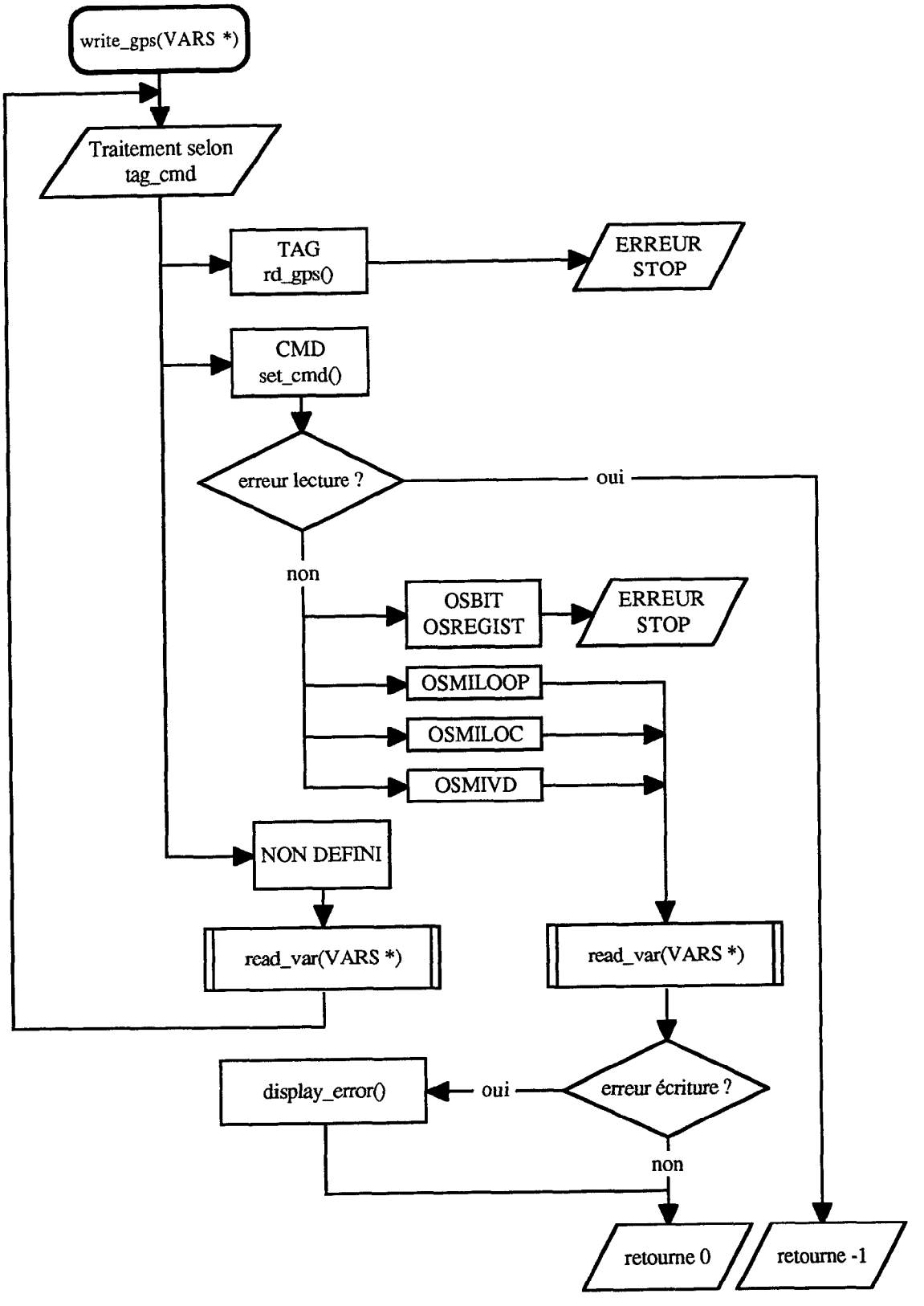

Figure 2.2.3./5 - Organigramme de la procédure writ_gps()

II.3. CONTRÔLE DU RÉACTEUR

La structure de contrôle a été transposée en langage C pour donner un programme de contrôle en temps réel. La chronologie des opérations réalisées est décrite sur la figure 2.3./1 Les fonctions d'acquisition (lecture) et d'écriture des données sont décrites précédemment. L'ensemble des variables à acquérir, et à écrire est déclaré dans deux procédures spécifiques, afin de permettre un accès par bloc. Un certain nombre d'exceptions figurent dans ces procédures pour prendre en compte les cas particuliers, comme par exemple, la calibration de l'analyseur de nitrate (impossibilité de lecture de la concentration en nitrate durant la calibration).

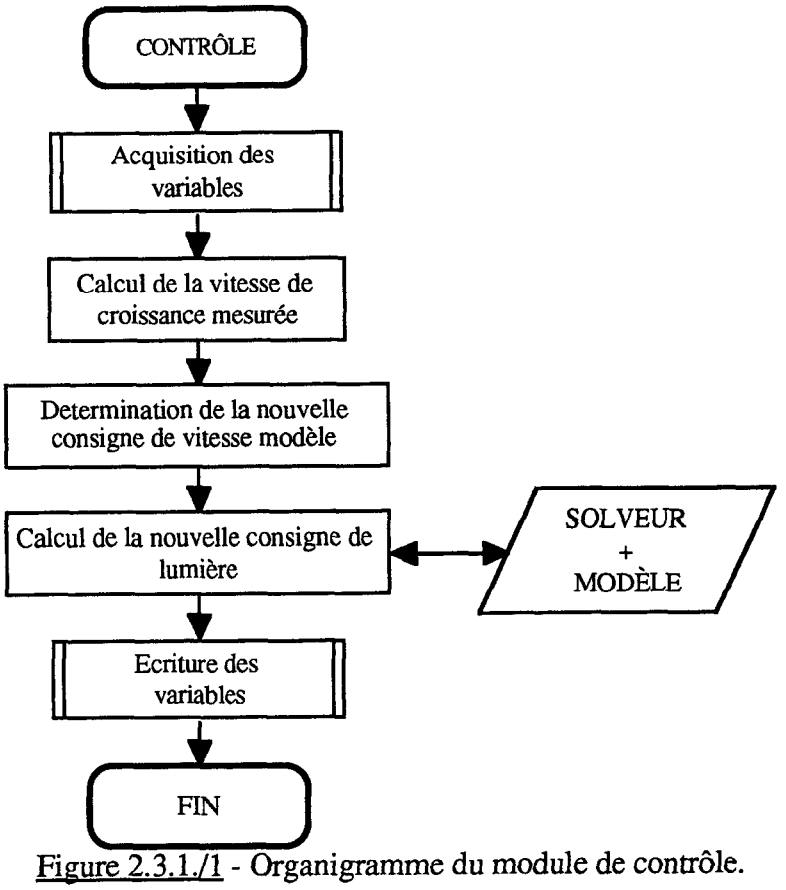

Figure 2.3.1./1 - Organigramme du module de contrôle.

II.3.1. CALCUL DE LA VITESSE DE CROISSANCE

Le principe du calcul est détaillé dans la TN 18.2., les lignes qui suivent présentent la mise en œuvre logicielle.

Une procédure spécifique permet de calculer la pente d'une variable de type compteur, selon la méthode des moindres carrés. Cette procédure nécessite deux paramètres, le nom de la variable (compteur) et le temps de mesure (60mn).

Les échantillons permettant le calcul de la pente sont contenus dans le tableau "val[]" de la *structure* "VARS" (associée au compteur). Une première phase analyse ces échantillons, pour tenir compte de la remise à zéro (RAZ) journalière du compteur. Si une RAZ est détectée dans l'intervalle de mesure, une correction des échantillons est réalisée pour permettre le calcul de la pente.

et la relation $Rxa = D\ Cxa = \frac{d}{V} Cxa$ permet de calculer la vitesse de croissance volumétrique.

II.3.2. DÉTERMINATION DE LA CONSIGNE EN VITESSE DU MODÈLE

Une fois la vitesse de croissance mesurée, l'erreur par rapport à la consigne est calculée ainsi que son intégrale. L'utilisation de la relation $s(t) = K_p\ \varepsilon(t) + K_i \int_0^t \varepsilon(t)\ dt$ détermine l'action de sortie du régulateur PI. Cette action ajoutée à la sortie du filtre de compensation donne la vitesse de croissance que le modèle doit utiliser pour calculer la consigne en lumière.

II.3.3. CALCUL DE LA CONSIGNE EN LUMIÈRE

II.3.3.a SOLVEUR

La détermination de la consigne en lumière est essentiellement réalisée à l'aide du modèle de connaissance. Le solveur recherche la valeur de Eb, donnant la vitesse de croissance demandée, avec les conditions courantes de la culture (température, concentration en biomasse, intensité lumineuse, etc...).

Les paramètres transmis au modèle sont regroupés dans une *structure* "REACT", qui constitue une représentation de l'état du réacteur.

Cette structure contient les variables nécessaires au modèle pour fonctionner :

nom	description
C_{xa}	Concentration en biomasse
C_{NO^3}	Concentration en Nitrate
temp	Température de la culture
Eb	Intensité mesurée par le capteur

en retour le modèle met à jour les champs suivants :

nom	description
Fr	Flux lumineux incident sur les parois du réacteur
Rxa	Vitesse volumétrique de croissance de la biomasse active
Rn	Vitesse volumétrique de consommation des Nitrate

La recherche d'une valeur de Eb (Ebr) conduisant à la vitesse de croissance désirée se fait en plusieurs étapes :

-1- détermination d'un intervalle où se trouve le point [Ebr,Rxar] recherché. Le premier point est pris à l'origine [Eb1,Rxa1] = [0,0], et le deuxième [Eb2,Rxa2] peut être pris égal au point courant [Ebc,Rxac], si celui-ci convient. Sans quoi, il suffit d'augmenter Eb2 et de calculer Rxa2 (avec le modèle) jusqu'à ce que le point supérieur soit obtenu.

-2- recherche de Ebr s'effectue en prenant le point médian de l'intervalle [Eb1,Eb2], ce qui donne deux intervalles. On conserve celui des deux qui contient Rxar et ainsi de suite jusqu'à l'obtention de Rxar± ε (ε erreur sur la vitesse, définie dans le programme).

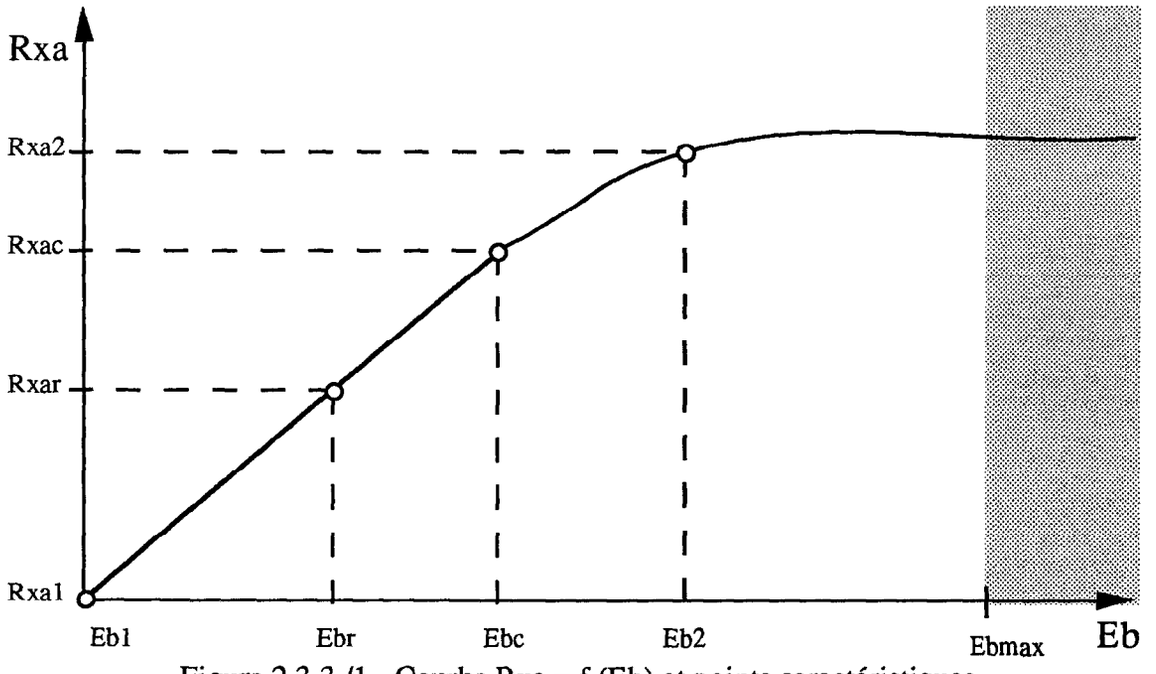

Figure 2.3.3./1 - Courbe Rxa = f (Eb) et points caractéristiques.

Le solveur utilise trois *structures* "REACT", deux pour les bornes de l'intervalle de recherche et une pour le point médian. La gestion de ces structures est réalisée avec des pointeur de façon à optimiser le temps de traitement .

Sous certaines conditions, le solveur peut avoir des problèmes de convergence (pas d'intégration choisit pour le modèle). Dans ce cas, lorsque l'intervalle de recherche [Eb1,Eb2] devient trop petit, Ebr prend alors la valeur moyenne de cet intervalle.

II.3.3.b MODÈLE DE CONNAISSANCE

La procédure, utilisant le modèle de connaissance, effectue successivement trois calculs :

- 1 - Calcul de α et δ, et détermination du flux incident F_R à partir de E_b, en utilisant la décomposition décrite dans la TN 18.2.

- 2 - Calcul du taux de croissance volumétrique moyen. L'énergie $4\pi J_Z$ disponible en chaque point du réacteur est obtenue avec la relation $\frac{4\pi J_r}{F_R} = \frac{1}{Z} \frac{2 \text{ ch } \delta Z}{\text{ch } \delta + \alpha \text{ sh } \delta}$. Les équations sont réécrites avec les considérations suivantes :

- passage en coordonnées réduites $z = r / R$

- prise en compte de l'existence des rayons R_2 et R'_2 définissant le volume actif, dans tous les cas possibles, en effectuant un test sur la valeur de $4\pi J_Z$. Le résultat de celui-ci autorise ou non l'intégration.

- incorporation de γ, rapport entre la surface active et la surface totale.

Le taux de croissance moyen s'écrit alors :

$$\langle \mu' \rangle = 2 \, \mu'_M \int_0^1 \frac{4\pi Jz}{K_J + 4\pi Jz} z \, (4\pi Jz > 1 W/m^2) \, dz$$

et le taux de croissance volumétrique moyen est donné par :

$$\langle R \rangle = \langle \mu' \rangle \, C_{PC} \, \rho$$

La difficulté est d'évaluer l'intégrale. Il existe différentes méthodes d'intégration numérique permettant de la cacluler.

Le paramètre à considérer dans le choix d'une méthode est le rapport précision / temps de calcul nécessaire. Il est certain que ce temps est loin d'être critique pour le contrôle de ce réacteur. Mais à terme, plusieurs modèles pourront être utilisés (un par compartiment). La méthode retenue est "la méthode du point milieu", qui donne pour un même nombre de pas d'intégration une meilleure approximation que "la méthode des trapèzes", tout en restant simple à mettre en œuvre.

La valeur de l'intégrale définie par cette méthode de f(x) sur un intervalle [a,b] est donnée par :

$$\int_a^b f(x)\,dx = h \sum_{i=0}^{n-1} f_{i+\frac{1}{2}} + \frac{1}{24} h^3 \sum_{i=0}^{n-1} f''(\xi i)$$

$$\text{avec } h = x_{i+1} - x_i = \frac{b-a}{n} \text{ et } x_i < \xi i < x_{i+1}$$

$h \sum_{i=0}^{n-1} f_{i+\frac{1}{2}}$ est une approximation de $\int_a^b f(x)\,dx$,

Il reste à déterminer le nombre "n" de pas d'intégration. Lorsque "n" tend vers l'infini, l'erreur tend vers zéro. Cependant, il n'est pas judicieux de choisir "n" trop élevé, car cela conduit à une augmentation du nombre d'itérations, et donc du temps de calcul. Il faut donc choisir une valeur donnant une précision et un temps de calcul acceptables.

Nous pouvons considérer qu'une précision de deux chiffres significatifs après la virgule pour la vitesse de croissance, est suffisante, étant donné les incertitudes sur les variables utilisées. La détermination de "n" a été réalisée en augmentant n jusqu'à ce que la vitesse soit proche (±0,01) de celle obtenue pour n très élevé (n > 1000).

Nous obtenons ainsi une valeur minimum de n qui se situe entre 50 et 100.

- 3 - Calcul de la vitesse de croissance (relations (4.13) à (4.19)), en fonction des concentrations minérales, et corrigées avec la température.

Le programme est compilé avec les options permettant de tirer le meilleur parti du microprocesseur utilisé, et en utilisant un processeur arithmétique. Ainsi, lors de l'exécution, le solveur associé au modèle nécessite environ deux à trois secondes pour trouver une solution (avec n=100).

III. CONCLUSION

Le programme décrit ci-avant a permi de valider la structure de contrôle sur un réacteur expérimental. Le système de contrôle INDUSTAR a montré sa capacité a accepter une application spécifique en interface avec un réseau d'échange. Il est important de souligner que tous les points qui aurait pu faire échouer l'étude ont été validés, et que de plus, ces validations permettent d'envisager l'extension au contrôle d'autre procédés sans limitations, en utilisant les briques de base existantes.

IV. ANNEXE

Liste et description des fichiers.

MELISSA.H	déclarations spécifiques
USERDEF.H	Déclarations procédures GPS
SPIRULIN.C	Fichier Main
MELFCT.C	Fonctions et base detemps
GPSFILE;C	Gestion des fichiers GPS
ALARMS.C	Getsion des alarmes et surveillance réseau
VARS.C	Ecriture et lecture des données
CONTROL.C	Programme de contrôle
CC.MAK	fichier MAKE
CC.BAT	Script pour la compilation complète avec MAKE
CPE.BAT	Script pour la compilation partielle
CPL.BAT	Script pour générer un fichier listing partiel
CP.BAT	Script pour compilation seule (partielle)
QL.BAT	Script pour l'édition d'un listing
QC.BAT	Script pour l'édition d'un fichier source
SAVE.BAT	Script pour la sauvegarde sur disquette de tous les fichiers source

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

  NAME          MELISSA.H

  AUTHOR        BINOIS C

  DESCRIPTION
    General Declarations

  UPDATES
    02-06-93

*****************************************************************/


/*--------------------------------
  constants for VARS
--------------------------------*/
#define TAG     128
#define CMD     255
#define UNDEF   0
#define NB_SAMP 0xFF        /* number of samples stored in val[]        */


/*--------------------------------
  structure for variables
--------------------------------*/

typedef struct _vars {
  char    name[9];          /* tag name                 */
  char    file[12];         /* file name                */
  int     type;             /* rd_gps/scl_cmd return code               */
```



```
  unsigned char tag_cmd;    /* TAG or CMD               */
  unsigned int dev_num;     /* controller number        */
  double  value;            /* current value            */
  int     i;                /* pointer on last value entered in val[] */
  double  val[NB_SAMP+1]; /* previous values            */
  double  min;
  double  max;
  double  sp;               /* set point for LOOP       */
  double  out;              /* out value for LOOP       */
  char    unit[5];          /* unit for analog values   */
  char    update;           /* ON when structure updated*/
} VARS;


/*--------------------------------
  structure for opened GPS files
--------------------------------*/

typedef struct _gps_file{
  char    file[15];             /* file name            */
  int     handler;              /* handler of gps file  */
  int     rank;                 /* rank of the gps file */
  struct  _gps_file   *next;    /* next opened gps file */
} GPS_FILE;


/*--------------------------------
  structure for reactor state
--------------------------------*/

typedef struct      _react{
  double  Cxa;
```

```
  double   Cno3;
  double   temp;
  double   press;
  double   Eb;
  double   Fr;
  double   rxa;
  double   m;
  double   ro2;
} REACT;

/*---------------------------------
  general constants
---------------------------------*/

#define TSAMP   60      /* sampling interval in secondes          */
#define SYNCHRO 1
#define ERROR_SPEED 0.01 /* max error for the model               */
#define VOLUME_LIGHT 3900
#define VOLUME_TOTAL 7000
#define CPT_CXA_UNIT 41.68

/*---------------------------------
  Mathematical constants
---------------------------------*/

#define PI          3.14159265359


/*---------------------------------
  colours
---------------------------------*/
```



```
#define BLACK   0
#define BLUE    1
#define GREEN   2
#define CYAN    3
#define RED     4
#define MAGENTA 5
#define BROWN   6
#define WHITE   7
```

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

NAME   USERDEF.H

AUTHORS (C) TOPTOOLS 1988,1989,1990

DESCRIPTION

  INDUSTAR General Purpose Station - User Include File

UPDATES
  90-05-10 - Add TiWay support

*************************************************************************/



/* ------------------------
  Miscellaneous constants definition
------------------------ */

#define ACTIV     1       /* command is active              */
#define INACTIV   0       /* command is inactive            */

#define ON        1       /* ON  value   for digital commands      */
#define OFF       0       /* OFF value   "       "       "     */



/* ------------------------
  rd_gps() return codes
------------------------ */

              /* Tag type              Structure   */
```



```
#define ISBIT     1       /* digital              IDIG  */
#define ISABUS    2       /* analog               IBUSA*/



/* ------------------------
  set_cmd() return codes
------------------------ */

              /* Cmd type              Structure   */
#define OSBIT     32      /* digital              ODIG */
#define OSMILOOP  41      /* analog   (Micon loop)     OMLO*/
#define OSREGUL   42      /* analog   (AB,TCS,PLS loop)  OREGU */
#define OSREGIST  52      /* analog   (register)       OREGIS */
#define OSMIVD    61      /* digital (Micon DV)        OMVD*/
#define OSMILOC   71      /* analog   (Micon loc)      OMLOC */



/* ------------------------
  gpserror values
------------------------ */

#define ERDOS     1       /* DOS problem              */
#define ENETW     2       /* network or mailbox       */
#define INVALID_FGPS 3    /* invalid GPS file         */
#define ETAGCMD   4       /* tag or command not found    */
#define EVARCOD   5       /* invalid variable codification    */
#define ECONV     6       /* floating point conversion      */
#define ETAGTYP   7       /* unknown tag type         */
#define EEQUIP    8       /* unknown PLC protocol        */
#define ECMDTYP   9       /* unknown command type        */
#define ENUMPLC   10      /* invalid device number       */
```

```
#define ERPROTECT11     /* protection key not found              */
#define ENSECTOR 12     /* invalid record number                 */
#define ETYPVARCAL 13     /* not a computed variable             */
#define EGDPREV  24     /* action on previous GD var not over    */
#define EGDTYPVAR 25    /* not a GD variable                     */
#define EGDPOLLIS 26    /* POLLIS.TAB not found                  */
#define EGDCMDFAIL 27     /* failed command to GD                */
#define ELOCAL   31     /* PLC in Local state (cmd impossible)   */
#define ISMODAUTO 32    /* AUTO mode : change is impossible      */
#define EPLSOVER  44    /* PLStar variable value out of limits   */



/* ------------------------
   Not selected command fields
------------------------ */

#define FVALNOTUSED0xF4240  /* 4bytes, not selected analog cmd field*/
#define IVALNOTUSED 64      /* 1byte,     not selected dig or loop fld.*/



/* ------------------------
   Specific definitions for Micon equipments
------------------------ */

#define MICLOCREM 151   /* local/remote                  */
#define MICCASCA 150    /* cascade                       */
#define MICAUTO  149    /* auto                          */
#define MICMANUAL 148   /* manual                        */
#define ISINCONFIG  155     /* Micon is starting configuration       */



/* ------------------------
```



```
   External declarations
------------------------ */


extern int gpserror;            /* variable to house error codes     */
extern int echonetw ;           /* network error code, when applicable   */
extern unsigned int nbtags;         /* # tags in activated group     */
extern unsigned int nbcommands;     /* # cmds in activated group     */
extern char *gettags () ;       /* tag or command name (8-char string)   */



/* ------------------------------------------------------------

                STRUCTURES FOR READING TAGS

------------------------------------------------------------ */



/* ------------------------
   Common header structure (tags and commands)
------------------------ */

typedef struct _iohead      {

   char     tag[9];        /* tag or cmd                    */
   char     tag_name[31];      /*  "       "  name          */
   unsigned    char tag_type;      /* tag or cmd type :
                    = 1  digital input
                    = 2  analog      "
                    = 3  digital output
                    = 4  analog      "      (loop)
                    = 5    "         "      (register)
```

```
                = 6    "          "     Micon  VD
                = 7    "          "     Micon  LOC
                ..................................                    */

  unsigned char   zone;    /* INDUSTAR C&C Station number      */
  unsigned int    dev_num; /* controller number                */
  unsigned char   dev_type; /* controller type :
                = 1  MICON
                = 2  JBUS-MODBUS
                = 3  STRUTHERS & DUNN
                = 4  ALLEN BRADLEY
                = 5  UNITELWAY
                = 8  TCS 6000
                = 9  TIWAY
                = 13 LAC
                = 14 PLStar TT
                = 15 Ghost Device TT
                ..................................                    */

  unsigned char   log_can;   /*   logical channel number          */

} IOHEAD;


/* ------------------------
   Digital Input
------------------------ */

typedef struct _idig {

  unsigned char val_state;          /* tag state      (0/1)          */
  unsigned char act_state;          /* active state                  */
```



```
  char        alarm_tag[9];       /* associated alarm tag          */
  char        fault_tag[9];       /* associated fault tag          */
  char        progress_tag[9];      /* associated in progress tag      */

  char        var_nam[7] ;      /* TIWAY - process variable name     */
  int         var_typ ;         /* TIWAY - process variable type     */
  int         var_num ;         /* TIWAY - process variable #        */

} IDIG;


/* ------------------------
   Analog tag
------------------------ */

typedef struct _ibusa {

  float       val;            /* value    (IEEE floating point)    */
  float       scale;          /* scale                             */
  float       vl;           /* very low limit                      */
  float       l;            /* low limit                           */
  float       h;            /* high limit                          */
  float       vh;           /* very high limit                     */
  char        unit[5];          /* unit                            */

  char        var_nam[7] ;      /* TIWAY - process variable name     */
  int         var_typ ;         /* TIWAY - process variable type     */
  int         var_num ;         /* TIWAY - process variable #        */

} IBUSA;
```

```
/* ----------------------
   Digital or Analog tag (without the header structure)
---------------------- */

typedef union _u_inp {

   IDIG   d;              /* when digital              */
   IBUSAbus;              /* when analog               */

} U_INP;


/* ----------------------
   Structure for the calls to rd_gps()
---------------------- */

typedef struct _igps {

   IOHEAD h;              /* common header structure   */
   U_INP i;               /* according to tag type and PLC  */

} IGPS;


/* ------------------------------------------------------

            STRUTURES FOR GETTING COMMAND INFORMATION

------------------------------------------------------ */
```



```
/* ----------------------
   Digital command
---------------------- */

typedef struct _odig {

   char   cmd_tag[9];       /* tag name associated to the command     */
   char   st_cmd;           /* tag    state (0/1)                     */
   char   as_cmd;           /* active state                           */
   char   cmd_file;         /* file number for ALLEN BRADLEY          */
   int    cmd_typ ;         /* TIWAY                                  */

   char   plcloc_tag[9];      /* local/remote tag                     */
   char   st_plcloc;        /* local(0)/remote(1) tag state           */
   char   as_plcloc;        /* active state                           */

   char   alarm_tag[9];     /* alarm tag associated to the command    */
   char   st_alarm;         /* state alarm tag (0/1)                  */
   char   as_alarm;         /* active state                           */

   char   fault_tag[9];     /* fault tag                              */
   char   st_fault;         /* state fault tag (0/1)                  */
   char   as_fault;         /* active state                           */

   char   instart_tag[9];     /* in progress tag                      */
   char   st_instart;       /* state in progress tag (0/1)            */
   char   as_instart;       /* active state                           */

   char   inhalt_tag[9];      /* in halt tag                          */
   char   st_inhalt;        /* state in halt tag (0/1)                */
   char   as_inhalt;        /* active state                           */
```

```
  char    localcmd_tag[9];      /* local command tag (element)          */
  char    st_localcmd;          /* state local command tag (0/1)        */
  char    as_localcmd;          /* active state                         */

  char    cmdtype;              /* command type = 0,1,2,3               */
  char    mult_tab;             /* not used by the G.P.S.               */

  unsigned devon_adr;           /* PLC bit state adresse ON             */
  char    as_devon;             /* active state for ON                  */
  char    mask_on;              /* bit mask for ALLEN BRADLEY           */

  unsigned devoff_adr;          /* PLC bit state addresse OFF           */
  char    as_devoff;            /* active state for OFF                 */
  char    mask_off;             /* bit mask for ALLEN BRADLEY           */

} ODIG;


/* ------------------------
  Analog command (register)
------------------------ */

typedef struct _oregis {

  char    cmd_tag[9];           /* tag name associated to the command   */
  float   val;                  /* tag measure value                    */
  int     cmd_typ ;             /* TIWAY                                */

  unsigned char cmd_mod;        /* mode variable PLStar                 */
  char    locrem_tag[9];        /* tag for local/remote state           */
  char    state;                /* PLC state Local(0)/Remote(1)         */
```



```
  char    as_locrem;            /* active state Local/Remote            */

  char    reg1_tag[9];          /* tag register 1                       */
  float   reg1_val;             /* value register 1                     */
  float   reg1_min;             /* value mini    reg. 1                 */
  float   reg1_max;             /* value maxi    reg. 1                 */
  char    reg1_unit[5];         /* unit register 1                      */
  unsigned reg1_adrhx;          /* PLC adress (hexa) of register 1      */
  unsigned char reg1_mod;       /* mode variable PLStar                 */
  char    reg1_file;            /* # file AB                            */
  int     reg1_typ ;            /* TIWAY                                */
  int     reg1_num ;            /* TIWAY                                */

  char    reg2_tag[9];          /* tag register 2                       */
  float   reg2_val;             /* value register 2                     */
  float   reg2_min;             /* value mini    reg. 2                 */
  float   reg2_max;             /* value maxi    reg. 2                 */
  char    reg2_unit[5];         /* unit reg. 2                          */
  unsigned reg2_adrhx;          /* PLC adress (hexa) of register 2      */
  unsigned char reg2_mod;       /* mode variable PLStar                 */
  char    reg2_file;            /* # file AB                            */
  int     reg2_typ ;            /* TIWAY                                */
  int     reg2_num ;            /* TIWAY                                */

  char    reg3_tag[9];          /* tag register 3                       */
  float   reg3_val;             /* value register 3                     */
  float   reg3_min;             /* valeur mini reg. 3                   */
  float   reg3_max;             /* valeur maxi reg. 3                   */
  char    reg3_unit[5];         /* unit register 3                      */
  unsigned reg3_adrhx;          /* PLC adress (hexa) of register 3      */
  unsigned char reg3_mod;       /* mode variable PLStar                 */
```

```
  char      reg3_file;        /* # file AB                */
  int       reg3_typ ;        /* TIWAY                    */
  int       reg3_num ;        /* TIWAY                    */

} OREGIS;



/* ----------------------
  Analog command (loop)
---------------------- */

typedef struct _oregu {

  char      cmd_tag[9];       /* tag name associated to the command     */
  float     val;          /* tag measure value              */
  int       cmd_typ ;         /* TIWAY                    */
  int       tiloopnb ;        /* TIWAY                    */
  char      cmd_mod;      /* mode variable PLStar           */

  char      spt_tag[9];       /* tag for setpoint               */
  float     spt_val;      /* setpoint value             */
  float     spt_min;      /* value mini     spt             */
  float     spt_max;      /* value maxi     spt             */
  char      spt_unit[5];      /* unit for setpoint              */
  unsigned spt_adr;       /* adresse ALLEN B            */
  unsigned spt_file;          /*   file number of spt value (A-B)       */
  char      spt_mod;      /* mode variable PLStar (cf. PLStar)      */
  int       spt_typ ;         /* TIWAY                    */
  int       spt_num ;         /* TIWAY                    */



  char      mo_tag[9];        /* tag for Manual Output          */
```



```
  float     mo_val;       /* M.O. value             */
  float     mo_min;       /* value mini     M.O.            */
  float     mo_max;       /* value maxi     M.O.            */
  char      mo_unit[5];       /* unit     M.O.              */
  unsigned mo_adr;        /* adresse ALLEN B            */
  unsigned mo_file;       /* file number of M.O.            */
  char      mo_mod;       /* mode variable PLStar (cf.PLStar)       */
  int       mo_typ ;      /* TIWAY                  */
  int       mo_num ;      /* TIWAY                  */

  char      stalo_tag[9];       /* tag for loop status            */
  char      stalo_mod;      /* status loop value              */
  char      stalo_unit[5];      /* unit for loop status           */
  unsigned stalo_adr;       /* adresse ALLEN B            */
  unsigned stalo_file;      /* file number of loop status         */
  char      stat_mod;       /* mode variable PLStar (cf. PLStar)      */
  int       stalo_typ ;     /* TIWAY                  */
  int       stalo_num ;     /* TIWAY                  */

} OREGU;



/* ------------------------
  Analog command (Micon loop)
------------------------ */



typedef struct _omlo {

  char      regutag[9];       /* tag associated to the command      */
  float     val;          /* current value read in the mailbox      */
```

```
  int     nloop;          /* loop number                  */
  float   sp;             /* setpoint value               */
  float   spmin;          /*   "      minimum value       */
  float   spmax;          /*   "      maximum   "         */
  char    spunit[5];      /*   unit for setpoint          */
  float   out;            /* outpoint value               */
  float   outmin;         /*   "      minimum value       */
  float   outmax;         /*   "      maximum   "         */
  char    outunit[5];     /* unit for outpoint            */
  float   ratio;          /* ratio value                  */
  float   ratiomin;       /*   "      mini                */
  float   ratiomax;       /*   "      maxi                */
  char    ratiounit[5];   /*   unit for ratio             */
  float   bias;           /* bias value                   */
  float   biasmin;        /*   "      mini                */
  float   biasmax;        /*   "      maxi                */
  char    biasunit[5];    /*   unit for bias              */
  unsigned char state;    /* loop state (auto/manual/cascade)  */

} OMLO;


/* ------------------------
  Analog command (Micon LOC)
------------------------ */

typedef struct _omloc {

  char    loctag[9];      /* tag name associated to the command  */
  float   val;            /* current LOC value            */
  int     nloc;           /* LOC number                   */
  float   locmin;         /*   minimum LOC value          */
```



```
  float   locmax;         /*   maximum "     "            */
  char    locunit[5];     /*   unit for LOC               */

} OMLOC;


/* ------------------------
  Digital command (Micon DV)
------------------------ */

typedef struct _omvd {

  char    vdtag[9];       /* tag name associated to the command  */
  int     val;            /* current Discrete Virtual value      */
  int     nvd;            /* Discrete Virtual number             */

} OMVD;


/* ------------------------
  Digital or Analog command (without the header)
------------------------ */

typedef union _u_outp {

  ODIG d;                 /* digital (common for all PLCs)       */
  OMLOml;                 /* Loop: MICON                  */
  OREGU l;                /* Loop: ALLEN-BRADLEY, PLStar, TIWAY */
  OREGIS r;               /* Register: MODBUS, JBUS, LAC, PLStar  */
                          /* ...ALLEN-B, UNITELWAY, TIWAY  */
  OMVDvd;                 /* MICON VD                     */
```

```
  OMLOC loc;          /* MICON LOC                */

} U_OUTP;


/* ----------------------
  Command structure for the calls to set_cmd() and wr_gps()
---------------------- */

typedef struct _ogps {

  IOHEAD h;           /* common header structure  */
  U_OUTP o;           /* according to the PLC     */

} OGPS;


/* ------------------------------------------------

                STRUCTURES TO SEND COMMANDS

------------------------------------------------ */


/* ----------------------
  Sending a command to JBUS, MODBUS, LAC equipments
---------------------- */

typedef struct _usjbus {

  char   status_bit;      /* cmd state switch ACTIV/INACTIV (default)   */
  char   action_bit;      /* new bit value (ON / OFF)                   */
```



```
  char   status_reg1;     /* cmd reg.1 switch ACTIV/INACTIV (default)   */
  float  newreg1;         /* new  register 1 value                      */

  char   status_reg2;     /* cmd reg. 2 switch ACTIV/INACTIV            */
  float  newreg2;         /* new  register 2 value                      */

  char   status_reg3;     /* cmd reg. 3 switch ACTIV/INACTIV            */
  float  newreg3;         /* new register 3 value                       */

} USJBUS;


/* ----------------------
  Send a command to a Micon equipment
---------------------- */

typedef struct _usmloop {

  char   status_sp;       /* cmd setpoint switch ACTIV/INACTIV(default) */
  float  val_sp;          /* new setpoint value                         */

  char   status_out;      /* cmd outpoint switch ACTIV/INACTIV          */
  float  val_out;         /* new outpoint value                         */

  char   status_ra;       /* cmd ratio switch ACTIV/INACTIV             */
  float  val_ra;          /* new ratio value                            */

  char   status_bi;       /* cmd bias switch ACTIV/INACTIV              */
  float  val_bi;          /* new bias value                             */
```

```
  char     status_sta;        /* cmd state switch ACTIV/INACTIV     */
  unsigned char mode;         /* new channel state value - use above MICON
                      definitions (MICLOCREM etc.)        */

  char     status_loc;        /* cmd LOC switch ACTIV/INACTIV     */
  float    val_loc;           /* new LOC value                */

  char     status_vd;         /* cmd DV switch ACTIV/INACTIV      */
  char     val_vd;            /* new DV value   ON/OFF          */

} USMLOOP;



/* ------------------------
  Send a command to :
  Allen-Bradley, PLStar, UnitelWay, Reliance, TCS or TIWAY equipment
------------------------ */

typedef struct _usalb {

  char     status_sp1;        /* cmd setpoint or register 1 switch ACTIV/INACTIV(default)          */
  float    val_sp1;           /* new setpoint or register 1 value           */

  char     status_out2;       /* cmd outpoint or register 2 switch ACTIV/INACTIV        */
  float    val_out2;          /* new outpoint or register 2 value           */

  char     status_sta3;       /* cmd loop state or register 3 switch ACTIV/INACTIV        */
  float    val_sta3;          /* new loop state value or register 3         */

  char     status_bit;        /* cmd state switch ACTIV/INACTIV     */
  char     val_bit;           /* new bit value      ON/OFF          */
```



```
} USALB;



/* ------------------------
  Send a command to the Mailbox
------------------------ */

typedef struct _uscalc {

  char     status_bit;        /* state switch (ACTIV/INACTIV) for new bit value       */
  char     val_bit;           /* new bit value                */

  char     status_num;        /* state switch (ACTIV/INACTIV) for new analog value        */
  float    val_num;           /* new analog value             */

  char     destable[32];          /* list of zones to send the message          */

} USCALC;



/* ------------------------
  Structure for the calls to wr_gps()
------------------------ */

typedef union _s_user {

  USJBUS jb;     /* MODBUS,JBUS,LAC                  */
  USMLOOP mic;/* MICON                          */
  USALB ab;   /* ALLEN BRADLEY,PLStar,UNITELWAY,TCS,RELIANCE,TIWAY*/
  USCALC cal;    /* Ghost Device TT                          */
```

```
} S_USER;



/* ------------------------
   GPS functions declarations
------------------------ */

extern    int open_gr(char *name_gr);
extern    int activ_gr(int h_gr);
extern    int rd_gps(char *mtag,struct _igps *_igps);
extern    int set_cmd(char *nom,struct _ogps *ogps);
extern    int wr_gps(struct _ogps *stout,union _s_user *stuser);
extern    int close_gr(int h_gr);
extern    char *gettags(unsigned int ii);
extern    int garde(unsigned int n,unsigned int nsecs);
```

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

NAME            SPIRULIN.C

AUTHOR          BINOIS C

DESCRIPTION

   MAIN PROGRAM listing file

UPDATES

   03-05-93

*****************************************************************/

#include <graph.h>
#include <process.h>
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>

#include "userdef.h"
#include "melissa.h"

int         my_interrupt();
char        buffer[100];
GPS_FILE *gps, *open_file_gps(), *activ_grp_gps();
int pointer;
VARS    essai;

main()

   {

   void wait_time(int);
   char        chr;
   int flag;

/*-------------------------------
```



```
   screen initialisation

-------------------------------*/

   screen_init();

/*-------------------------------

   set interruption

-------------------------------*/

   if(signal(SIGINT,my_interrupt)==(int(*)())-1)

      {

      _outtext("\nCouldn't set SIGINT *** Program Terminated ***\n");
      exit(0);
      }

/*-------------------------------

   timebase synchronisation

-------------------------------*/

   wait_time(1);

/*-------------------------------

   open groups and active one

-------------------------------*/

   gps=open_file_gps();
   gps=activ_grp_gps();
   set_gps(gps);

/*-------------------------------

   variables initialisation

-------------------------------*/

   init_vars();
```

```
  init_alarm();

/*-----------------------------------

  waiting loop

-----------------------------------*/


  do

    {

    if(!timebase())

      {

      control_spiru();
      flag=0;
      }

    else

      {

      if(!flag)

        {

        alarm_spiru();
        check_network();
        flag=1;
        }

      }

    if(kbhit())

      {

      chr=getch();
      }

    }

  while(1);
  }


/*-----------------------------------

  interruption of programm

-----------------------------------*/


int       my_interrupt()

  {
```



```
char      ch;
signal(SIGINT,SIG_IGN);
_outtext("Terminate processing ? ");
ch=getch();
if((ch=='y')||(ch=='Y'))

   {

   close_grp_gps();
   _outtext("\nbye.... *** Program Terminated by user ***\n");
   wait_time(5);
   _setvideomode(_DEFAULTMODE);
   exit(0);
   }

if(signal(SIGINT,my_interrupt)==(int(*)())-1)

   {

   _outtext("\nCouldn't set SIGINT *** Program Terminated ***\n");
   exit(0);
   }

_outtext("Continue...\n");
return;
}
```

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

  NAME              MELFCT.C

  AUTHOR            BINOIS C

  DESCRIPTION

     general functions listing file

  UPDATES

     10-03-93

*********************************************************************/


#include <graph.h>

#include <time.h>

#include <stdio.h>

#include <stdlib.h>

#include <process.h>

#include <string.h>

#include <malloc.h>

#include <math.h>


#include "melissa.h"

#include "userdef.h"


/*---------------------------------

   time base generator

---------------------------------*/


timebase()

   {

   static    unsigned long lastsamp = 0; /* last sampling */

   static    unsigned tsamp = TSAMP ;      /* sampling interval */

   static    unsigned long last_display;
   time_t    ltime;
   char      buffer[100];
   struct    tm   *dt;
   double    minute;
```



```
time(&ltime);
if(last_display<ltime)

   {

   dt=localtime(&ltime);
   dt->tm_mon++;
   sprintf(buffer," %02d/%02d/%02d          %02d:%02d:%02d",dt->tm_mday,
   dt->tm_mon,dt->tm_year,dt->tm_hour,dt->tm_min,dt->tm_sec);
   display_time(buffer);
   last_display=ltime;
   }



time(&ltime);
if (lastsamp==0)

   {

   display_status("timebase synchronisation ...");
   minute=TSAMP/60;
   while((ceil(dt->tm_min/minute)!=dt->tm_min/minute)||(dt->tm_sec!=0))

      {

      time(&ltime);
      dt=localtime(&ltime);
      dt->tm_mon++;
      if(last_display<ltime)

         {

         sprintf(buffer," %02d/%02d/%02d          %02d:%02d:%02d",dt->tm_mday,

         dt->tm_mon,dt->tm_year,dt->tm_hour,dt->tm_min,dt->tm_sec);
         display_time(buffer);
         last_display=ltime;
         }

      }

   lastsamp=ltime;
   display_status(" ");
   return(0);
   }

if ((lastsamp+tsamp)<=ltime)

   {

   lastsamp=lastsamp+tsamp;
   return(0);
   }

return(ltime);
}
```

```
/*-------------------------------

  wait i seconds

-------------------------------*/


wait_time(int i)

  {

  char        buffer[100];
  unsigned long start_time;

  while(timebase()==0)

     {

     }

  start_time=timebase();
  while(timebase()<start_time+i)

     {

     }

  }
```

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

  NAME              GPSFILE.C

  AUTHOR            BINOIS C

  DESCRIPTION

     management of gps files

  UPDATES

     25-03-93

*****************************************************************/


#include <malloc.h>

#include <process.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>


#include "melissa.h"

#include "userdef.h"


static GPS_FILE *gps;

/*-----------------------------------

  open all gps files declared in GPS.FIL

-----------------------------------*/


GPS_FILE *open_file_gps()

  {

  FILE     *fp;
  int      hl,i,file_rank;
  char     buffer[80],str[80];
  GPS_FILE     *gp,*gp_save;
  int      file_opened;


  file_opened=OFF;
  file_rank=1;
  display_status("opening GPS files ...");
  fp=fopen("GPS.FIL","r");
  if(fp==NULL)
```



```
    {

    display_error("Could not open file GPS.FIL ... *** program stopped ***");
    exit(0);
    }

  while(fscanf(fp,"%s",buffer)!=EOF)

    {

    if(sscanf(buffer,"BASE:%s",str)!=1)

      {

      display_error("Syntax error in file GPS.FIL ... *** program stopped ***");
      fclose(fp);
      exit(0);
      }

    for(i=1;;i++)

      {

      sprintf(buffer,"%s%02d.GPS",str,i);
      hl=open_gr(buffer);
      if(hl==-1)

        {

        break;
        }

      gp=(GPS_FILE *)malloc(sizeof(GPS_FILE));
      if(gp==NULL)

        {

        sprintf(buffer,"Can't allocate memory for %s%02d.GPS *** program stopped ***",str,i);
        display_error(buffer);
        exit(0);
        }

      if(!file_opened)

        {

        gp->next=gp;
        gp_save=gp;
        }

      sprintf(gp->file,"%s",buffer);
      gp->handler=hl;
      gp->rank=file_rank;
      gp->next=gp_save->next;
      gp_save->next=gp;
      gp_save=gp;
      file_opened=ON;
      file_rank++;
      sprintf(buffer,"file %s opened ...\n",gp->file);
      _outtext(buffer);
      wait_time(1);
      }
```

```
    }
  if(!file_opened)
    {
    display_error("No GPS file found ... *** program terminated ***");
    exit(0);
    }
  fclose(fp);
  gps=gp;
  return(gp);
  }


/*-------------------------------
  active one group using GPS_FILE struct
-------------------------------*/


GPS_FILE *activ_grp_gps()


  {
  GPS_FILE    *gp;
  char        buffer[80];
  int         ret_activ_gr;


  gp=gps->next;
  ret_activ_gr=activ_gr(gp->handler);
  if(ret_activ_gr==-1)
    {
    sprintf(buffer,"Can't activate file %s ... *** program stopped ***",gps->file);
    display_error(buffer);
    exit(0);
    }
  gps=gp;
  display_activ_group(gps);
  return(gp);
  }



/*-------------------------------
  close all groups using GPS_FILE struct
-------------------------------*/
```



```
void close_grp_gps()


  {
  GPS_FILE    *gp;
  char        buffer[80];
  int         file_rank, ret_close, all_closed;


  file_rank=gps->rank;
  all_closed=OFF;
  while(!all_closed)
    {
    ret_close=close_gr(gps->handler);
    if(ret_close==-1)
      {
      sprintf(buffer,"\nCan't close file %s ...\n",gps->file);
      display_error(buffer);
      error_gps();
      }
    else
      {
      sprintf(buffer,"\nFile %s closed",gps->file);
      _outtext(buffer);
      wait_time(1);
      }
    gps=gps->next;
    if(gps->rank==file_rank)
      {
      all_closed=ON;
      }
    }
  display_no_activ_group();
  }
```

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


NAME            SCREEN.C


AUTHOR          BINOIS C


DESCRIPTION

   screen and graphic functions listing file


UPDATES

   06-05-93


*****************************************************************/



#include <stdio.h>

#include <graph.h>

#include "melissa.h"



/*-----------------------------------

   screen initialisation

-----------------------------------*/


void screen_init(void)

   {

   _setvideomode(_TEXTC80);
   _setbkcolor((long)RED);
   _settextwindow(1,1,25,80);
   _clearscreen(_GWINDOW);


   _settextcolor(BLACK);
   tab(3,19);
   _outtext("messages ...");
```



```
   _setbkcolor((long)BLUE);
   _settextcolor(WHITE);
   tab(26,2);
   _outtext("      ***  M E L I S S A ***  ");


   _settextwindow(4,2,18,79);
   _clearscreen(_GWINDOW);
   _settextwindow(20,2,24,79);
   _clearscreen(_GWINDOW);


   use_message_window();
   _wrapon(_GWRAPON);
   _displaycursor(_GCURSOROFF);


   }


/*-----------------------------------

   display activ group

-----------------------------------*/



display_activ_group(GPS_FILE *gps)

   {

   char     buffer[100];
   struct   rccoord txtpos;
   txtpos=_gettextposition();


   use_group_window();
   sprintf(buffer,"ACTIVE GROUP : %s",gps->file);
   _outtext(buffer);


   use_message_window();
   _settextposition(txtpos.row,txtpos.col);
   }


/*-----------------------------------

   display no activ group

-----------------------------------*/



display_no_activ_group()

   {

   char     buffer[100];
   struct   rccoord txtpos;
```

```
  txtpos=_gettextposition();


  use_group_window();
  sprintf(buffer,"ACTIVE GROUP : --------.GPS");
  _outtext(buffer);


  use_message_window();
  _settextposition(txtpos.row,txtpos.col);
  }




/*---------------------------------

  display current time

---------------------------------*/


display_time(char *buffer)

  {

  struct   rccoord txtpos;
  txtpos=_gettextposition();


  use_time_window();
  _outtext(buffer);


  use_message_window();
  _settextposition(txtpos.row,txtpos.col);
  }


/*---------------------------------

  display result in main window

---------------------------------*/


display_result(char *buffer,short x,short y)

  {

  struct   rccoord txtpos;


  if((x>76)||(y>12))

    {

    display_error("Can't display result : coordinates error on :");
    display_error(buffer);
```



```
    return(-1);
    }

  txtpos=_gettextposition();


  use_display_window();
  _settextposition(y,x);
  _outtext(buffer);


  use_message_window();
  _settextposition(txtpos.row,txtpos.col);
  }




/*---------------------------------

  display system status

---------------------------------*/


display_status(char *buffer)

  {

  struct   rccoord txtpos;
  txtpos=_gettextposition();


  use_status_window();
  _outtext(buffer);


  use_message_window();
  _settextposition(txtpos.row,txtpos.col);
  }




/*---------------------------------

  display error messages

---------------------------------*/


display_error(char *buffer)

  {

  _setbkcolor((long)GREEN);
  _settextcolor(RED+16);
  _outtext(buffer);
  printf("\a\a\a");
```

```
   _setbkcolor((long)BLUE);
   _settextcolor(WHITE);
   _wrapon(_GWRAPON);
   _displaycursor(_GCURSOROFF);
   _outtext("\n");
   }


/*-----------------------------------

   use messages area

-----------------------------------*/


use_message_window()

   {

   _settextwindow(20,3,24,78);
   _setbkcolor((long)BLUE);
   _settextcolor(WHITE);
   _wrapon(_GWRAPON);
   _displaycursor(_GCURSOROFF);
   }


/*-----------------------------------

   use group display area

-----------------------------------*/


use_group_window()

   {

   _settextwindow(4,3,4,39);
   tab(3,4);
   _settextcolor(WHITE);
   _setbkcolor((long)BLUE);
   _clearscreen(_GWINDOW);
   }


/*-----------------------------------

   use time display area

-----------------------------------*/


use_time_window()

   {
```



```
   _settextwindow(4,58,4,79);
   tab(58,4);
   _settextcolor(WHITE);
   _setbkcolor((long)BLUE);
   }


/*-----------------------------------

   use status display area

-----------------------------------*/


use_status_window()

   {

   _settextwindow(18,3,18,78);
   tab(3,18);
   _settextcolor(WHITE);
   _setbkcolor((long)BLUE);
   _clearscreen(_GWINDOW);
   }


/*-----------------------------------

   use main display area

-----------------------------------*/


use_display_window()

   {

   _settextwindow(6,3,17,78);
   _settextcolor(WHITE);
   _setbkcolor((long)BLUE);
   tab(3,6);
   }


/*-----------------------------------

   move the cursor to the position (x,y)

-----------------------------------*/


tab(short x,short y)

   {

   _settextposition(y,x);
   }
```

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

  NAME          ALARMS.C

  AUTHOR        BINOIS C

  DESCRIPTION

    ALARM management listing file

  UPDATES

    05-05-93

*********************************************************************/

#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#include "userdef.h"
#include "melissa.h"

/*-----------------------------------

  variables declarations

-----------------------------------*/

VARS    raz_alarm1;         /* RAZ alarm P100 1          */
VARS    raz_alarm2;         /* RAZ alarm P100 2          */
VARS    raz_alarm3;         /* RAZ alarm P100 3          */
VARS    raz_alarm4;         /* RAZ alarm P100 4          */

VARS    raz_main_alarm;
VARS    raz_micon1_alarm;
VARS    raz_pressure;
VARS    activ_lamp;
VARS    alarm_micon1;
```



```
VARS    alarm_micon1_value;
VARS    level_switch;

VARS    speed,biomass;

char buffer[100];

/*-----------------------------------

  alarm programm

-----------------------------------*/


void alarm_spiru()

  {

  acq_alarm();
  return(0);
  if(alarm_micon1.value==1)

    {

    if(alarm_micon1_value.value&&1)

      {

      /* alarm on p100 no 1 */

      }

    if(alarm_micon1_value.value&&2)

      {

      /* alarm on p100 no 2 */

      }

    if(alarm_micon1_value.value&&4)

      {

      /* alarm on p100 no 3 */

      }

    if(alarm_micon1_value.value&&8)

      {

      /* alarm on p100 no 4 */

      }

    }

  if(speed.value>50)
```

```
    {
    if(level_switch.value)
      {
      wait_time(15);
      read_var(&level_switch);
      if(level_switch.value)
        {
        activ_lamp.sp=ON;
        display_error(" check harvesting, a problem has been detected ... \n");
        }
      }
    }
  if(biomass.value>(biomass.sp+10))
    {
    if(!biomass.out)
      {
      activ_lamp.sp=ON;
      display_error(" check level detectors, a problem has been detected ...\n");
      }
    }

  send_alarm();
  }

/*-----------------------------------

  alarms initialisation

-----------------------------------*/

init_alarm()
  {
  display_status("Initialisation of ALARMS ...");
  sprintf(raz_alarm1.name,"VD--0141");
  sprintf(raz_alarm2.name,"VD--0145");
  sprintf(raz_alarm3.name,"VD--0149");
  sprintf(raz_alarm4.name,"VD--0153");
  sprintf(raz_main_alarm.name,"VD--0155");
  sprintf(raz_micon1_alarm.name,"VD--0156");
  sprintf(raz_pressure.name,"VD--0144");
  sprintf(activ_lamp.name,"VD--0146");
```



```
  sprintf(alarm_micon1.name,"LOC-0122");
  sprintf(alarm_micon1_value.name,"LOC-0121");
  sprintf(level_switch.name,"DI--0136");

  sprintf(speed.name,"LOC-0130");
  sprintf(biomass.name,"LOOP0107");

  wait_time(1);
  display_status(" ");
  }

/*-----------------------------------

  alarms acquisition

-----------------------------------*/

acq_alarm()
  {
  display_status("Acquisition of ALARMS ...");

  read_var(&raz_alarm1);
  read_var(&raz_alarm2);
  read_var(&raz_alarm3);
  read_var(&raz_alarm4);
  read_var(&raz_main_alarm);
  read_var(&raz_micon1_alarm);
  read_var(&raz_pressure);
  read_var(&activ_lamp);
  read_var(&alarm_micon1);
  read_var(&alarm_micon1_value);
  read_var(&level_switch);

  read_var(&speed);
  read_var(&biomass);

  wait_time(1);
  display_status(" ");
  }

/*-----------------------------------

  alarms updating

-----------------------------------*/

send_alarm()
```

```
{
display_status("Updating ALARMS ...");


write_var(&raz_alarm1);
write_var(&raz_alarm2);
write_var(&raz_alarm3);
write_var(&raz_alarm4);
write_var(&raz_main_alarm);
write_var(&raz_micon1_alarm);
write_var(&raz_pressure);
write_var(activ_lamp);


wait_time(1);
display_status(" ");
}
```

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

  NAME              ALARMS.C

  AUTHOR            BINOIS C

  DESCRIPTION
    ALARM management listing file

  UPDATES
    05-05-93

*****************************************************************/

#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#include "userdef.h"
#include "melissa.h"

/*-------------------------------
   variables declarations
-------------------------------*/



char buffer[100];

/*-------------------------------
   alarm programm
-------------------------------*/
```



```
void alarm_spiru()

   {

   acq_alarm();
   send_alarm();
   }


/*-------------------------------
   alarms initialisation
-------------------------------*/


init_alarm()

   {

   display_status("Initialisation of ALARMS ...");
   wait_time(1);
   display_status(" ");
   }


/*-------------------------------
   alarms acquisition
-------------------------------*/


acq_alarm()

   {

   display_status("Acquisition of ALARMS ...");


   wait_time(1);
   display_status(" ");
   }


/*-------------------------------
   alarms updating
-------------------------------*/


send_alarm()

   {

   display_status("Updating ALARMS ...");
```

```
wait_time(1);
display_status("  ");
}
```

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

  NAME              VARS.C

  AUTHOR            BINOIS C

  DESCRIPTION

     access to data via gps functions

     and variables management

  UPDATES

     03-05-93

*****************************************************************/



#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <process.h>
#include <string.h>
#include <malloc.h>

#include "melissa.h"
#include "userdef.h"

static GPS_FILE *gps;
GPS_FILE *activ_grp_gps();
int my_interrupt();

/*-------------------------------

   acquisition of gps pointer

-------------------------------*/
```



```
void set_gps(GPS_FILE *gp)

  {

  gps=gp;
  }



/*-------------------------------

   read function

-------------------------------*/


void read_var(VARS *read_var)


  {

  double    value;
  double    read_gps(VARS *);
  int       file_rank;
  char      buffer[80];


  value=read_gps(read_var);
  if(value==-1)

    {

    file_rank=gps->rank;
    while(value==-1)

      {

      gps=activ_grp_gps();
      value=read_gps(read_var);
      if(file_rank==gps->rank)

        {

        sprintf(buffer,"Can not find %s in gps files",read_var->name);
        display_error(buffer);
        my_interrupt();
        }

      }

    }

read_var->i++;
read_var->i&=NB_SAMP;
read_var->val[read_var->i]=value;
read_var->value=value;
}
```

```
/*--------------------------------

  read gps sub_function

--------------------------------*/


double read_gps(VARS *read_vars)


  {

  OGPS    read_ogps;
  IGPS    read_igps;
  int     read_count,j,k;
  int     ret_code,tag_found;
  char    buffer[9],name[9];
  double  read_value;


  read_value=0;
  for(read_count=0;read_count<1;read_count++)

  {

  switch(read_vars->tag_cmd)

    {

    case TAG:

      {

      ret_code=rd_gps(read_vars->name,&read_igps);
      if(ret_code==-1)

        {

        error_gps();
        return(-1);
        }

      switch(ret_code)

        {

        case ISBIT:

          {

          if(read_igps.i.d.val_state==ACTIV)

            {read_value=1;}

          else

            {read_value=0;}
```



```
          if(!read_vars->update)

            {

            if(read_igps.i.d.act_state==ACTIV)

              {read_vars->max=1;}

            else

              {read_vars->max=0;}

            read_vars->dev_num=read_igps.h.dev_num;
            sprintf(read_vars->file,"%s",gps->file);
            read_vars->update=ON;
            }

          break;
          }

        case ISABUS:

          {

          read_value=read_igps.i.bus.val;
          if(!read_vars->update)

            {

            read_vars->min=read_igps.i.bus.l;
            read_vars->max=read_igps.i.bus.h;
            sprintf(read_vars->unit,"%s",read_igps.i.bus.unit);
            read_vars->dev_num=read_igps.h.dev_num;
            sprintf(read_vars->file,"%s",gps->file);
            read_vars->update=ON;
            }

          break;
          }

        }

      break;
      }

    case CMD:

      {

      ret_code=set_cmd(read_vars->name,&read_ogps);
      if(ret_code==-1)

        {

        error_gps();
        return(-1);
        }

      switch(ret_code)

        {

        case OSBIT:
```

```
case OSREGIST:

  {

  sprintf(buffer,"%s is not a valid tag name for MICON ..."

  ,read_vars->name);
  display_error(buffer);
  break;
  }

case OSMILOOP:

  {

  read_value=read_ogps.o.ml.val;
  read_vars->out=read_ogps.o.ml.out;
  read_vars->sp=read_ogps.o.ml.sp;
  if(!read_vars->update)

    {

    read_vars->min=read_ogps.o.ml.spmin;
    read_vars->max=read_ogps.o.ml.spmax;
    sprintf(read_vars->unit,"%s",read_ogps.o.ml.spunit);
    read_vars->dev_num=read_ogps.h.dev_num;
    sprintf(read_vars->file,"%s",gps->file);
    read_vars->update=ON;
    }

  break;
  }

case OSMILOC:

  {

  read_value=read_ogps.o.loc.val;
  read_vars->sp=read_ogps.o.loc.val;
  if(!read_vars->update)

    {

    read_vars->min=read_ogps.o.loc.locmin;
    read_vars->max=read_ogps.o.loc.locmax;
    sprintf(read_vars->unit,"%s",read_ogps.o.loc.locunit);
    read_vars->dev_num=read_ogps.h.dev_num;
    sprintf(read_vars->file,"%s",gps->file);
    read_vars->update=ON;
    }

  break;
  }

case OSMIVD:

  {

  read_value=read_ogps.o.vd.val;
  read_vars->sp=read_ogps.o.vd.val;
  if(!read_vars->update)
```



```
    {

    read_vars->dev_num=read_ogps.h.dev_num;
    sprintf(read_vars->file,"%s",gps->file);
    read_vars->update=ON;
    }

  break;
  }

  }

break;
}

default:

  {

  tag_found=OFF;
  j=nbtags+nbcommands;
  for(k=0;k<j;k++)

    {

    sprintf(buffer,"%s",gettags(k));
    if(strcmp(buffer,read_vars->name)==0)

      {

      tag_found=ON;
      if(k<nbtags)

        {

        ret_code=rd_gps(read_vars->name,&read_igps);
        read_vars->tag_cmd=TAG;
        }

      else

        {

        ret_code=set_cmd(read_vars->name,&read_ogps);
        read_vars->tag_cmd=CMD;
        }

      if(ret_code==-1)

        {

        error_gps();
        return(-1);
        }

      else

        {

        read_vars->type=ret_code;
        read_count--;
        }
```

```
            break;
            }

          }

        if(!tag_found)

          {

          return(-1);
          }

        }

      }

    }

    if(read_value==-1)

      {

      return(-0.9999999);
      }

    return(read_value);
    }


/*-----------------------------------

    write function

-----------------------------------*/


void

write_var(VARS *write_var)


    {

    int       file_rank,ret_code;
    char      buffer[80];


    ret_code=write_gps(write_var);
    if(ret_code==-1)

      {

      file_rank=gps->rank;
      while(ret_code==-1)

        {

        gps=activ_grp_gps();
        ret_code=write_gps(write_var);
        if(file_rank==gps->rank)
```



```
          {

          sprintf(buffer,"Can not find %s in gps files",write_var->name);
          display_error(buffer);
          my_interrupt();
          break;
          }

        }

      }

    }




/*-----------------------------------

    write gps sub function

-----------------------------------*/


int write_gps(VARS *write_vars)


    {

    OGPS     write_ogps;
    S_USER   write_s_user;
    int      ret_code,tag_written;
    time_t   ltime;
    char     buffer[80];


    tag_written=OFF;
    while(!tag_written)

    {

    switch(write_vars->tag_cmd)

      {

      case TAG:

        {

        sprintf(buffer,"Can't write the TAG %s ...",write_vars->name);
        display_error(buffer);
        my_interrupt();
        break;
        }

      case CMD:
```

```
{

ret_code=set_cmd(write_vars->name,&write_ogps);
if(ret_code==-1)

  {

  error_gps();
  return(-1);
  }

switch(ret_code)

  {

  case OSBIT:

  case OSREGIST:

    {

    sprintf(buffer,"%s is not a valid tag name for MICON ..."

    ,write_vars->name);
    display_error(buffer);
    break;
    }

  case OSMILOOP:

    {

    if((write_ogps.o.ml.state&16)==16)

      {

      write_s_user.mic.mode=MICLOCREM;
      write_s_user.mic.status_sta=ACTIV;
      if(wr_gps(&write_ogps,&write_s_user)==-1)

        {

        error_gps();
        }

      break;
      }

    if((write_ogps.o.ml.state&1)==0)

      {

      write_s_user.mic.mode=MICAUTO;
      write_s_user.mic.status_sta=ACTIV;
      if(wr_gps(&write_ogps,&write_s_user)==-1)

        {

        error_gps();
        }

      break;
      }
```



```
      write_s_user.mic.status_sta=INACTIV;
      write_s_user.mic.status_out=INACTIV;
      write_s_user.mic.status_ra=INACTIV;
      write_s_user.mic.status_bi=INACTIV;
      write_s_user.mic.status_loc=INACTIV;
      write_s_user.mic.status_vd=INACTIV;
      write_s_user.mic.val_sp=write_vars->sp;
      write_s_user.mic.status_sp=ACTIV;
      tag_written=ON;
      break;
      }

    case OSMILOC:

      {

      write_s_user.mic.val_loc=write_vars->sp;
      write_s_user.mic.status_loc=ACTIV;
      tag_written=ON;
      break;
      }

    case OSMIVD:

      {

      write_s_user.mic.val_vd=(int)write_vars->sp;
      write_s_user.mic.status_vd=ACTIV;
      tag_written=ON;
      break;
      }

    }

  break;
  }

default:

  {

  read_var(write_vars);
  }

}

}

if(wr_gps(&write_ogps,&write_s_user)==-1)

  {

  error_gps();
  time(&ltime);
  sprintf(buffer,"When writing %s at time %s",write_vars->name,

  ctime(&ltime));
  display_error(buffer);
  return(0);
  }

else
```

```
    {
    return(0);
    }
  }


/*-----------------------------------
  errors management
-----------------------------------*/


error_gps()
  {
  switch (gpserror)
    {
    case ERDOS:
      {
      display_error("A DOS problem has occured ...\n");
      my_interrupt();
      break;
      }
    case ENETW:
      {
      display_error("Network or Mailbox fault ...\n");
      break;
      }
    case INVALID_FGPS:
      {
      display_error("Incorrect GPS file ...\n");
      my_interrupt();
      break;
      }
    case ETAGCMD:
      {
      break;
      }
    case ETAGTYP:
    case ECMDTYP:
```



```
      {
      display_error("Unknown CMD or TAG ...");
      my_interrupt();
      break;
      }
    case ECONV:
      {
      display_error("Floating point conversion error ...");
      my_interrupt();
      break;
      }
    case EEQUIP:
      {
      display_error("Unknown PLC protocol ...");
      my_interrupt();
      break;
      }
    case ENUMPLC:
      {
      display_error("Invalid device number ...");
      my_interrupt();
      break;
      }
    default:
      {
      display_error("Unidentified error ...");
      my_interrupt();
      break;
      }
    }
  }


/*-----------------------------------
  check if mailbox is refresh
-----------------------------------*/


check_network()
  {
  display_status("Checking Network ...");
  while(garde(101,1))
```

```
    {

    }

display_status(" ");
}
```

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

  NAME          CONTROL.C

  AUTHOR        BINOIS C

  DESCRIPTION
    CONTROL PROGRAM listing file

  UPDATES
    10-03-93

*****************************************************************/

#include <malloc.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#include "userdef.h"
#include "melissa.h"

#define SPEED_TIME 60  /* time for measuring growth speed */

int my_interrupt();

/*-----------------------------------
  variables declarations
-----------------------------------*/

VARS   cxa;           /* biomasse concentration          */
VARS   cpt_cxa;       /* biomass regulation counter      */
```



```
VARS   Rxa;           /* growth speed of biomasse        */
VARS   Rxa_set_point; /*                                 */

VARS   nitrate;       /* nitrate concentration           */
VARS   cpt_nitrate;   /* nitrate regulation counter      */
VARS   Rnitrate;      /* nitrate consumption rate        */
VARS   cal_nitrate;   /* nitrate calibration switch      */

VARS   light;         /* light intensity in the reactor  */
VARS   temperature;   /* temperature in the reactor      */
VARS   pH;            /* pH of culture                   */
VARS   pressure;      /* gaseous pressure in the reactor */

VARS   Rxa_model;     /* growth speed calculated by model */
VARS   Rxa_error;     /*   error between model and measure */
VARS   error_average;
VARS   error_sigma;
VARS   sum_error;
VARS   cpt_cxa_unit;

VARS   Kp,Ki;
VARS   rxa_filtered;

REACT  air_lift;

char   buffer[100];

/*-----------------------------------
  control programm
-----------------------------------*/


void control_spiru()
  {
  acq_vars();
  calc_rx();
  air_lift.Cxa=cxa.value;
  air_lift.Cno3=nitrate.value;
  air_lift.temp=temperature.value;
```

```
  air_lift.press=pressure.value;
  air_lift.Eb=light.value;
  model(&air_lift);


  filter();
  Rxa_model.sp=rxa_filtered.sp

        +Kp.value*Rxa_error.sp+Ki.value*sum_error.sp;
  solver(&air_lift,Rxa_model.sp);
  light.sp=air_lift.Eb;
  Rxa_model.sp=air_lift.rxa;


  send_vars();
  result();
  }




/*----------------------------------

  mathematical model

----------------------------------*/



model(REACT *react)

  {

  double   zpc=.135;
  double   zp=.57;
  double   zch=0.0085;
  double   zg=0;
  double   za;
  double   Ea=871;
  double   Es=167;
  double   alpha,delta,delta3;
  double   Fr,Fr1,Fr2,Fr3;
  double   R,R1,R2,R3,Rb;
  double   z;
  double   jstep=0.01;
  double   pij,pijz;
  double   Kj=20;
  double   KN=5.3;
  double   muM=0.54;
  double   yn=0.42;


  double   coeft,coefN,Rmean;
  R=0.048;
  R1=0.0302;
  R2=0.02585;
  R3=0.0115;
  Rb=0.0095;


/*   general parameters ----------------------------------*/
```



```
  za=zpc+zch;
  alpha=sqrt(za*Ea/(za*Ea+(1+zg)*Es));
  delta=(za*Ea+(1+zg)*Es)*react->Cxa/1000*alpha*R;
  delta3=delta*R2/R;


/*   incident flux determination ------------------------*/



  Fr3=react->Eb*Rb/PI/R3;
  z=R3/R2;
  Fr2=Fr3*z/(2*alpha)*(cosh(delta3)+alpha*sinh(delta3))/sinh(delta3*z);
  Fr1=Fr2*R2/R1;
  z=R1/R;
  Fr=Fr1*z/(2*alpha)*(cosh(delta)+alpha*sinh(delta))/sinh(delta*z);
  react->Fr=Fr;


/*   determination of the mean growth rate --------------*/



  pij=0;
  for(z=jstep/2;z<=1-jstep/2;z+=jstep)

    {

    if((z<R2/R)||(z>R1/R))

      {

      pijz=Fr/z*2*cosh(delta*z)/(cosh(delta)+alpha*sinh(delta));
      if(pijz>=1)

        {

        pij+=2*z*pijz/(Kj+pijz)*jstep;
        }

      }

    }

  Rmean=muM*pij*zpc*react->Cxa*VOLUME_LIGHT/VOLUME_TOTAL;


/* temperature and nitrates correction */



  coeft=0.8*exp(-pow((react->temp-35)/10,2))+0.2;
  if(react->Cno3<20)

    {

    react->Cno3=100;
    }

  coefN=react->Cno3/(KN+react->Cno3);
coefN=1;
```

```
  react->rxa=Rmean*coefN*coefL;
  react->rn=yn*react->rxa;
  }




/*-----------------------------------

  solver using model

-----------------------------------*/



solver(REACT *react,double Rx_seek)

  {

  char      buffer[100];
  REACT   *react1,*react2,*react_seek,*preact;


  display_status("Model working ...");
  model(react);
  react->rxa=(react->rxa<0.2)? 0.2 : react->rxa;
  Rx_seek=(Rx_seek<=0.2)? 0.2 : Rx_seek;
  if(fabs(react->rxa-Rx_seek)<ERROR_SPEED)

    {

    display_status(" ");
    return(0);
    }

  react1=malloc(sizeof(REACT));
  react2=malloc(sizeof(REACT));
  react_seek=malloc(sizeof(REACT));


  copy_react(react,react1);
  copy_react(react,react2);
  copy_react(react,react_seek);
  react1->Eb=0;
  react1->rxa=0;
  while(Rx_seek>react2->rxa)

    {

    react2->Eb+=50;
    if(react2->Eb>100)

      {

      react2->Eb=100;
      model(react2);
      Rx_seek=react2->rxa-0.2;
      break;
      }

    model(react2);
    }
```



```
  while(fabs(react_seek->rxa-Rx_seek)>ERROR_SPEED)

    {

    react_seek->Eb=(react1->Eb+react2->Eb)/2;
    model(react_seek);
    if(Rx_seek>react_seek->rxa)

      {

      preact=react1;
      react1=react_seek;
      react_seek=preact;
      }

    else

      {

      preact=react2;
      react2=react_seek;
      react_seek=preact;
      }

    if(fabs(react1->Eb-react2->Eb)<0.001)

      {

      break;
      }

    }

  copy_react(react_seek,react);
  free(react_seek);
  free(react1);
  free(react2);
  display_status(" ");
  }




/*-----------------------------------

  copy structure reacta to reactb

-----------------------------------*/



copy_react(REACT *reacta,REACT *reactb)



  {

  reactb->Cxa=reacta->Cxa;
  reactb->Cno3=reacta->Cno3;
  reactb->temp=reacta->temp;
  reactb->press=reacta->press;
```

```
  reactb->Eb=reacta->Eb;
  reactb->Fr=reacta->Fr;
  reactb->rxa=reacta->rxa;
  reactb->rn=reacta->rn;
  reactb->ro2=reacta->ro2;
  }



/*-----------------------------------

  growth speed calculation

-----------------------------------*/


calc_rx()

  {

  double   diff_cpt(VARS *,int);
  double   average_var(VARS *,int);
  double   average2_var(VARS *,int);
  double   slope_var(VARS *,int);
  double   slope_cpt(VARS *,int);
  double   dcxa_dt;



  dcxa_dt=slope_var(&cxa,SPEED_TIME);
  Rxa.sp=(slope_cpt(&cpt_cxa,SPEED_TIME)*cpt_cxa_unit.value/VOLUME_TOTAL

  *cxa.value)*60/SPEED_TIME;


  Rxa_error.sp=Rxa_set_point.value-Rxa.sp;
  sum_error.sp+=Rxa_error.sp;
  error_average.sp=average_var(&Rxa_error,240);
  error_sigma.sp=sqrt(average2_var(&Rxa_error,240)-pow(error_average.sp,2));
  }



/*-----------------------------------

  filter for the reference compensation

-----------------------------------*/


filter()
```



```
  {

  static double T[3];
  double p[3],q[3],k;


  k=0.0316227766;
  p[1]=1.96255627;
  p[2]=-0.96327226;
  q[0]=1;
  q[1]=-0.014618827;
  q[2]=0.0367277331;


  T[0]=k*Rxa_set_point.sp+p[1]*T[1]+p[2]*T[2];
  T[2]=T[1];
  T[1]=T[0];
  rxa_filtered.sp=k*Rxa_set_point.sp*q[0]+q[1]*T[1]+q[2]*T[2];
  }



/*-----------------------------------

  variables initialisation

-----------------------------------*/


init_vars()

  {

  REACT  init_react;
  double  delta;
  int     jj;


  display_status("Initialisation of variables ...");


/*  TAG and COMMAND name initialisation      */


  sprintf(cxa.name,"LOOP0107");
  sprintf(cpt_cxa.name,"LOC-0111");
  sprintf(Rxa.name,"LOC-0130");
  sprintf(Rxa_set_point.name,"LOC-0132");


  sprintf(nitrate.name,"LOOP0103");
  sprintf(cpt_nitrate.name,"LOC-0109");
  sprintf(Rnitrate.name,"LOC-0133");
  sprintf(cal_nitrate.name,"DI--0125");


  sprintf(light.name,"LOOP0105");
  sprintf(temperature.name,"LOOP0106");
  sprintf(pH.name,"LOOP0104");
```

```
  sprintf(pressure.name,"LOOP0102");


  sprintf(Rxa_model.name,"LOC-0131");
  sprintf(Rxa_error.name,"LOC-0133");
  sprintf(error_average.name,"LOC-0134");
  sprintf(error_sigma.name,"LOC-0135");
  sprintf(sum_error.name,"LOC-0136");
  sprintf(cpt_cxa_unit.name,"LOC-0137");
  sprintf(Kp.name,"LOC-0138");
  sprintf(Ki.name,"LOC-0139");
  sprintf(rxa_filtered.name,"LOC-0140");


/*   Variables and Counters initialisation          */

  acq_vars();
  init_react.Cxa=cxa.value;
  init_react.Cno3=nitrate.value;
  init_react.temp=temperature.value;
  init_react.press=pressure.value;
  init_react.Eb=light.value;
  model(&init_react);
  Rxa_set_point.sp=init_react.rxa;
  write_var(&Rxa_set_point);


  delta=-init_react.rxa/init_react.Cxa*VOLUME_TOTAL/cpt_cxa_unit.value

  *TSAMP/3600;
  fill_struct_cpt(&cpt_cxa,delta);
  fill_struct_cpt(&cpt_nitrate,0.42*delta);
  fill_struct_var(&cxa);


  Rxa_error.value=0;
  fill_struct_var(&Rxa_error);
  sum_error.value=0;
  fill_struct_var(&sum_error);
  fill_struct_var(&Kp);
  fill_struct_var(&Ki);


/* filter initialisation */


  for(jj=0;jj<500;jj++)

    {

    filter();
    }

  write_var(&rxa_filtered);


  wait_time(1);
  display_status(" ");
  }
```



```
/*-----------------------------------

   variables acquisition

-----------------------------------*/



acq_vars()

  {

  display_status("Acquisition of variables ...");


  read_var(&cxa);
  if(fabs(cxa.value-cxa.sp)>20)

    {

    cxa.value=cxa.sp;
    cxa.val[cxa.i]=cxa.sp;
    }

  read_var(&cpt_cxa);
  read_var(&Rxa);
  read_var(&Rxa_set_point);
  read_var(&cpt_cxa_unit);


/*   nitrate analyser calibration          */


  read_var(&cal_nitrate);
  if(!cal_nitrate.value)

    {

    read_var(&nitrate);
    read_var(&cpt_nitrate);
    }



  read_var(&Rnitrate);


  read_var(&light);
  read_var(&temperature);
  read_var(&pH);
  read_var(&pressure);


  read_var(&Rxa_model);
  read_var(&Rxa_error);
  read_var(&error_average);
  read_var(&error_sigma);
  read_var(&sum_error);
  read_var(&Ki);
  read_var(&Kp);
```

```
   read_var(&rxa_filtered);

   display_status(" ");
   }


/*-----------------------------------

   commands updating

-----------------------------------*/


send_vars()

   {

   display_status("Updating variables ...");


   write_var(&Rxa);
   write_var(&Rxa_set_point);


   write_var(&Rnitrate);
   write_var(&light);


   write_var(&Rxa_model);
   write_var(&Rxa_error);
   write_var(&error_average);
   write_var(&error_sigma);
   write_var(&sum_error);
   write_var(&rxa_filtered);
   display_status(" ");
   }


/*-----------------------------------

   prepare result of control for display

-----------------------------------*/


result()

   {

   void      display_result(char *,short,short);
   char buffer[150];


   sprintf(buffer,"Concentrations              Biomass");
   display_result(buffer,1,1);
   sprintf(buffer,"mg/l");
```



```
   display_result(buffer,32,1);
   display_result(buffer,32,2);
   sprintf(buffer,"Nitrate");
   display_result(buffer,17,2);
   sprintf(buffer,"%.1f",cxa.value);
   display_result(buffer,26,1);
   sprintf(buffer,"%.1f",nitrate.value);
   display_result(buffer,26,2);


   sprintf(buffer,"Light");
   display_result(buffer,1,4);
   sprintf(buffer,"Eb                  W/m2");
   display_result(buffer,22,4);
   sprintf(buffer,"Fr                  W/m2");
   display_result(buffer,22,5);
   sprintf(buffer,"%.1f",light.value);
   display_result(buffer,26,4);
   sprintf(buffer,"%.1f",air_lift.Fr);
   display_result(buffer,26,5);


   sprintf(buffer,"Kinetics                  measured              mg/lh");
   display_result(buffer,1,7);
   sprintf(buffer,"%.2f",Rxa.sp);
   display_result(buffer,26,7);


   sprintf(buffer,"set-point                  mg/lh");
   display_result(buffer,15,8);
   sprintf(buffer,"%.2f",Rxa_set_point.value);
   display_result(buffer,26,8);


   sprintf(buffer,"model                  mg/lh");
   display_result(buffer,19,9);
   sprintf(buffer,"%.2f",Rxa_model.sp);
   display_result(buffer,26,9);


   sprintf(buffer,"error                  mg/lh");
   display_result(buffer,19,10);
   sprintf(buffer,"%.2f",Rxa_error.sp);
   display_result(buffer,26,10);


   sprintf(buffer,"average error                  mg/lh");
   display_result(buffer,11,11);
   sprintf(buffer,"%.2f",error_average.sp);
   display_result(buffer,26,11);


   sprintf(buffer,"sigma error                  mg/lh");
   display_result(buffer,13,12);
   sprintf(buffer,"%.2f",error_sigma.sp);
   display_result(buffer,26,12);


   }
```

```
/*-----------------------------------

   calculate the delta count during time t in minutes

-----------------------------------*/


double diff_cpt(VARS *diff_var, int diff_time)

   {

   int j;
   int i_samp,i_prev,nb_samp;
   double total_count;


   total_count=0;
   nb_samp=ceil(diff_time*60/TSAMP);
   for(j=0;j<nb_samp;j++)

      {

      i_samp=(diff_var->i-j)&NB_SAMP;
      i_prev=(i_samp-1)&NB_SAMP;
      total_count+= ( diff_var->val[i_samp]>=diff_var->val[i_prev]) ?

      diff_var->val[i_samp]-diff_var->val[i_prev] : diff_var->val[i_samp];
      }

   return(total_count);
   }




/*-----------------------------------

   calculate the variable variation during time t in minutes

-----------------------------------*/


double diff_var(VARS *diff_var, int diff_time)

   {

   double dvar_dt;
   int nb_samp;


   nb_samp=ceil(diff_time*60/TSAMP);
   dvar_dt=diff_var->val[diff_var->i]-diff_var->val[(diff_var->i

   -nb_samp)&NB_SAMP];
   return(dvar_dt);
   }
```



```
/*-----------------------------------

   calculate the average during time t in minutes

-----------------------------------*/


double average_var(VARS *diff_var, int diff_time)

   {

   int j;
   int i_samp,nb_samp;
   double average;


   average=0;
   nb_samp=ceil(diff_time*60/TSAMP);
   for(j=0;j<nb_samp;j++)

      {

      i_samp=(diff_var->i-j)&NB_SAMP;
      average+=diff_var->val[i_samp];
      }

   average/=nb_samp;
   return(average);
   }



/*-----------------------------------

   calculate the average^2 during time t in minutes

-----------------------------------*/


double average2_var(VARS *diff_var, int diff_time)

   {

   int j;
   int i_samp,nb_samp;
   double average;


   average=0;
   nb_samp=ceil(diff_time*60/TSAMP);
   for(j=0;j<nb_samp;j++)

      {

      i_samp=(diff_var->i-j)&NB_SAMP;
      average+=pow(diff_var->val[i_samp],2);
      }

   average/=nb_samp;
   return(average);
   }
```

```
/*-----------------------------------

  fill val[i] with the current value

-----------------------------------*/


fill_struct_var(VARS *fill_struct)


  {
  int jj;

  for(jj=0;jj<=NB_SAMP;jj++)
    {
    fill_struct->val[jj]=fill_struct->value;
    }
  }



/*-----------------------------------

  fill val[i] with the current value and delta between each value

-----------------------------------*/


fill_struct_cpt(VARS *fill_struct,double _delta)


  {
  int jj,kk,ll;

  for(jj=0;jj<NB_SAMP;jj++)
    {
    kk=(fill_struct->i-jj)&NB_SAMP;
    ll=(kk-1)&NB_SAMP;
    fill_struct->val[ll]=fill_struct->val[kk]+_delta;
    }
  }
```



```
/*-----------------------------------

  calculate the slope of variable by the least mean square method

-----------------------------------*/


double slope_var(VARS *slope_var,int diff_time)

  {
  int ii,jj,kk;
  int nb_samp;
  double slope;
  double sumxi, sumyi, sumxiyi, sumxi2;


  sumxi=0;
  sumyi=0;
  sumxiyi=0;
  sumxi2=0;


  nb_samp=ceil(diff_time*60/TSAMP);
  for(ii=0;ii<nb_samp;ii++)
    {

    jj=slope_var->i-ii;
    kk=(slope_var->i-ii)&NB_SAMP;
    sumxi+=jj;
    sumyi+=slope_var->val[kk];
    sumxiyi+=jj*slope_var->val[kk];
    sumxi2+=pow((double)jj,2);
    }

  slope=nb_samp*(nb_samp*sumxiyi-sumxi*sumyi)/(nb_samp*sumxi2-sumxi*sumxi);
  return(slope);
  }



/*-----------------------------------

  calculate the slope of counter by the least mean square method

-----------------------------------*/


double slope_cpt(VARS *slope_cpt,int diff_time)

  {
  int ii,jj,kk,ll;
  int nb_samp;
  double slope;
  double sumxi, sumyi, sumxiyi, sumxi2;
```

```
double raz_cpt;


raz_cpt=0;
sumxi=0;
sumyi=0;
sumxiyi=0;
sumxi2=0;


nb_samp=ceil(diff_time*60/TSAMP);
for(ii=0;ii<nb_samp;ii++)

  {

  jj=slope_cpt->i-ii;
  kk=(slope_cpt->i-ii)&NB_SAMP;
  ll=(kk-1)&NB_SAMP;
  sumxi+=jj;
  sumyi+=(slope_cpt->val[kk]-raz_cpt);
  sumxiyi+=jj*(slope_cpt->val[kk]-raz_cpt);
  sumxi2+=pow((double)jj,2);
  if(slope_cpt->val[ll]>slope_cpt->val[kk])

    {

    raz_cpt=slope_cpt->val[ll];
    }

  }

slope=nb_samp*(nb_samp*sumxiyi-sumxi*sumyi)/(nb_samp*sumxi2-sumxi*sumxi);
return(slope);
}
```

```
# make the executable programm for spirulina compartement control

CC = CL
CFLAGS = /c /AL /Zp /FPi87 /Zi /G2 /Fs
LINKER = link
LFLAGS = /CO
FILES  = spirulin.c melfct.c gpsfile.c screen.c vars.c control.c alarms.c
OBJS  = spirulin.obj melfct.obj gpsfile.obj screen.obj vars.obj control.obj alarms.obj
LIBRARY = GPSL.LIB GRAPHICS.LIB
#executable file
BASE = GO

$(BASE).exe : melissa.h
   $(CC) $(CFLAGS) $(FILES)

spirulin.obj : spirulin.c
   $(CC) $(CFLAGS) $**

melfct.obj : melfct.c
   $(CC) $(CFLAGS) $**

gpsfile.obj : gpsfile.c
   $(CC) $(CFLAGS) $**

screen.obj : screen.c
   $(CC) $(CFLAGS) $**

vars.obj : vars.c
   $(CC) $(CFLAGS) $**

control.obj : control.c
   $(CC) $(CFLAGS) $**
```



```
alarms.obj : alarms.c
   $(CC) $(CFLAGS) $**

$(BASE).EXE : $(OBJS)
   $(LINKER) $(LFLAGS) $**,$*.EXE,$*.MAP,$(LIBRARY)
```

```
qc %1.lst
```

```
copy *.c a:

copy *.h a:

copy *.bat a:

copy *.mak a:

copy*.ini a:

dir a:
```

```
make cc.mak
```

```
CL /FcTEST /AL /Zp /FPi /Zi /Fs %1.C GPSL.LIB GRAPHICS.LIB
```

```
CL /FeGO /AL /Zp /G2 /FPi87 /Zi /Fs spirulin.c screen.c melfct.c GPSL.LIB GRAPHICS.LIB
```

```
CL %1 /AL /Zp /FPi;
if errorlevel 1 goto exit
LINK %1,%1,%1,\c\lib\cm+\c\lib\llibfp+\c\lib\llibc+\c\lib\libh+\c\lib\gpsl
:exit
```