Universitat Autònoma de Barcelona

MELISSA collaboration agreement ECT/FG/CB/95.205

ESTEC/CONTRACT11549/95/NL/FG

- TECHNICAL NOTE 25.5 -

General Purpose Station

main program update.

Version : 1
Issue : 0

PONS P.L. ; ALBIOL J. ; GODIA F.
Dpt. Enginyeria Química
Universitat Autònoma de Barcelona
08193 Bellaterra, Barcelona, Spain

SEPTEMBER 1996

0.- Introduction.

The operability of the Melissa loop relies on the development of a control system able to optimise the behaviour of its four linked subsystems. The progress in its implementation is done by developing different control modules that will have to be tested on the current implementation of the pilot plant. Such modules are going to be included as a part of a main program that manages different accessory tasks like the initialisation of the system, communication with the other software or the screen management.

At its lower level (level 0) each compartment is controlled by P-100 controllers (see Figure 0) , the management of this controllers is done by a PC running a comercial software (INDUSTAR) connected via RS-422 communicaton line. The P-100 controllers of the same compartment are communicated through a Vertical Communication Controller (VCC). The INDUSTAR software allows for the control, storage and graphical display of the data as well as modification of the P-100 working parameters. This computer software can be governed by another station (GPS) which allows a more complex treatment of the data. The last two stations are connected via an ARCNET. The GPS software allows the development of specific aplications implementing level 1 and 2 of the control strategy as well as optimisation and fault detection.

The first version of the GPS software, as described in TN 18.3, allowed only the control for the fourth compartment of the loop (Spirulina), displaying treated data in text mode. Relaying on the INDUSTAR software for data storage.

Evolution in the implementation of the loop required to upgrade the software so that it could control all the four compartments and perform a global control. The development of this upgrade follows the directions given on TN 28.2, provided by ADERSA, to modify the previous version of the program in the following points.

- Extend the system for all the compartments - screen, declarations and main loop - (according to TN 28.2 point 4.4.1).
- Set a reliable alarm system (according to TN 28.2 point 4.4.2).
- Implement a first order filter to reduce the measurement noise (according to TN 28.2 point 4.4.3).
- Disk storage (according to TN 28.2 point 4.4.4).

During software implementation other improvements were added to the program, like :

- Graphical data display.
- Help system.
- File management system allowing file copying, moving and removal.

- Modification of the network test routine to avoid the program to hang when the net is phisicaly disconnected.
- Modified GPS software version allowing for a several simultaneous monitoring stations.

This document describes the implementation of the described main program to which the different control modules will be incorporated as soon as they will be available. In the following more detailed description of the modifications can be found.

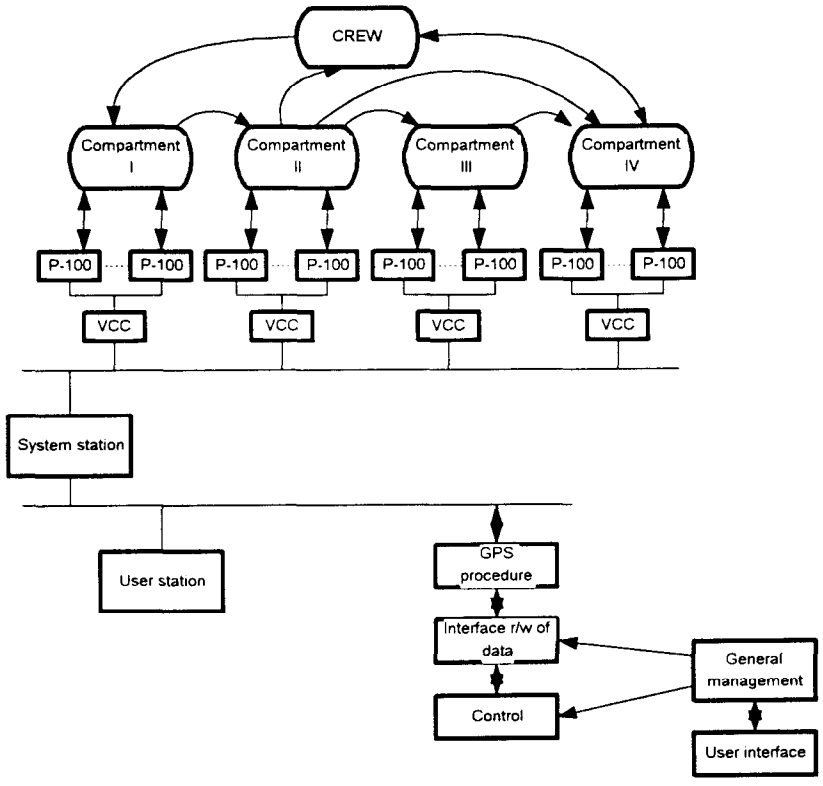


Figure 0

1.- General description.

1.1 .- Introduction

According to TN 28.2 the program should be modified to support the control implementation of the other compartments. The main loop and initialisation tasks are schematised in figure 1. This document will describe more extensively its differences with the previous version.

Generally the initial procedures consist in getting the system status, do a safety test and configure the GPS. After that, the variables are initialised and a time synchronisation is done before entering the main loop. The main loop (shown in Figure 1) mainly checks for a keyboard input and the main timer. The keyboard input allows for a screen change, graphical representation of data or file management. The main timer allows to schedule the network and file system status, the control process and the disk storage if necessary.

In the following, a description of the different involved modules will be done. This will be done in the same sequence as it takes place in the program, for a better description of its function and the new features included.

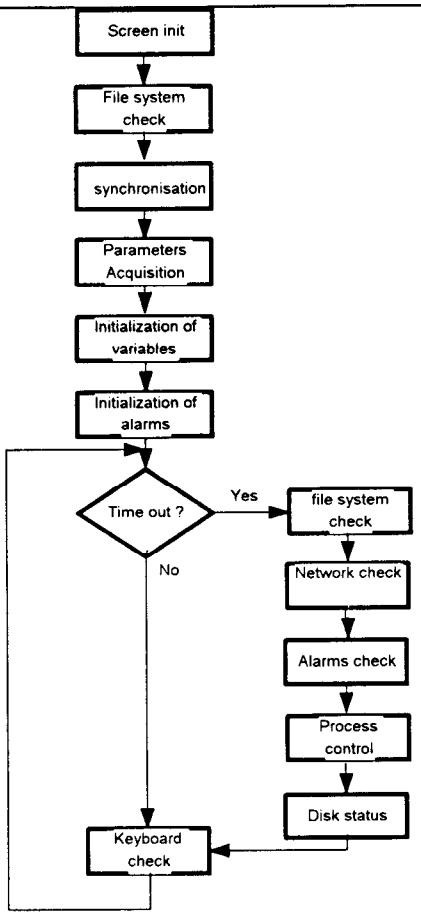


Figure 1

1.2 .- Screen init.

This is a simple module. Its task is to set the screen mode and draw the main screen.

1.3 .- File system check.

This is a new feature included for safety, its task is to monitorize the amount of free disk space left on the system and to avoid the loss of data. There are three basic states, normal working, warning, and alarm.

The file systems check routines have been developed so that, after testing the hard disk for free space an action is taken depending on the amount of free space left. If it is lower than 400 KB the GPS sends - every minute - a warning, but it keeps on storing data. If the free

space is lower than 100 KB, all storing data are disabled. A screen warning appears every minute.

In case of a lack of disk space, a procedure is required to be able to recover free space without stopping the GPS. For this, a menu for file managing has been implemented, where it is possible to copy, move and delete a restricted group of files. This point will be further developed in another chapter.

The objective of this module is check for free space and take measures.

1.4 .- Synchronisation

This module task is to synchronise the GPS internal timer with the second 0 of the real time clock.

1.5 .- Parameters acquisition

Many of the characteristics of the GPS, such as filters and timers, should be subjected to be changed for a better improvement of the GPS. In order to allow this possibility, a configuration file has been included. In this file all those features can be changed. In the case of a lack of this file, the parameters are going to be set in a default value.

In addendum 1 a detailed description of how to configure the GPS is done.

1.6 .- Initialisation of variables.

This module calls the functions to initialise the variables of each reactor.[1] When it is executed, all the variables are assigned to a name and read once. Its implementation has been done so as to allow an easy extension of the GPS for the new compartments.

1.7.- Initialisation of alarms.

The function of this module is similar to the last one, but this variables are used to check the alarms of the system.

[1] To initialise a variable is assign a name to the correponding field of the structure, and perform the first read on the net. For further details refer to the structure VARS, located in the file ***melissa.h***.

1.8 .- The main loop.

1.8.1 .- Check disk.

This file system check is the same that is made at the beginning of the execution of the GPS.

1.8.2 .- Network check.

In this part the correct operation of the network is checked. It is executed immediately prior to the use of the network.

1.8.3.- Alarms management.

Due to the fact that the GPS will be working for long periods of time, it is very important that it can control all the error cases, warn about them and take an action according to the type of error, if possible. The alarm module branches in four local modules, one for each compartment. For the time being the only one that has functionality is the one for the *Spirulina* compartment. This is the one that is going to be described.

There are two kind of possible alarms. Those that are given by the P100, and those that are located in a higher control level. The former group is located in the P100 modules of the next diagram, and the later is in the module labelled as "analysis". In any case when an alarm is set, an error message will appear on the screen, and a signal, to turn the alarm off in the P100, will be sent.

The algorithm used is shown in Figure 2.

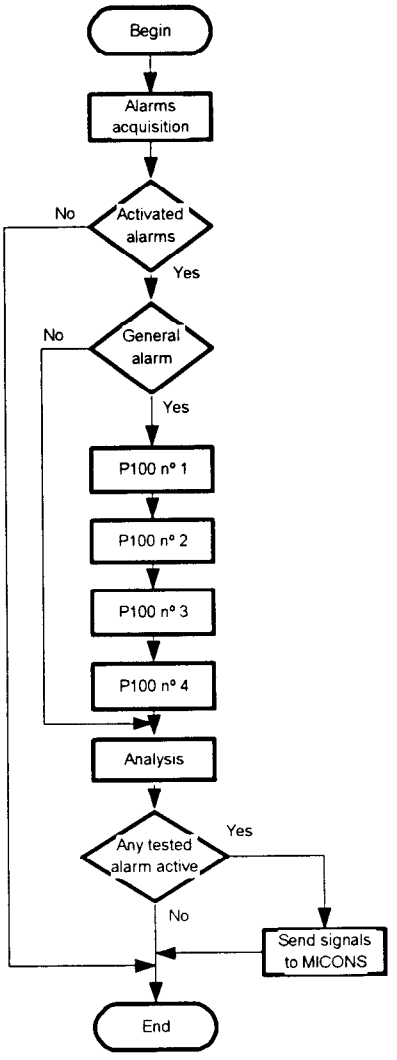


Figure 2

1.8.4.- Process control

This is the most important module for controlling the compartments, It is in charge of executing the control routines each time the corresponding timer reaches zero. This implementation is easy to expand and simple to program.

The decision to change some parameters for the reactor, is going to be taken in the control routines that are called by this module (see Figure 3) When that decision is taken a

bitmap[2] is going to be the output of the module, the values of the output variables of this module are going to be stored by the next module.

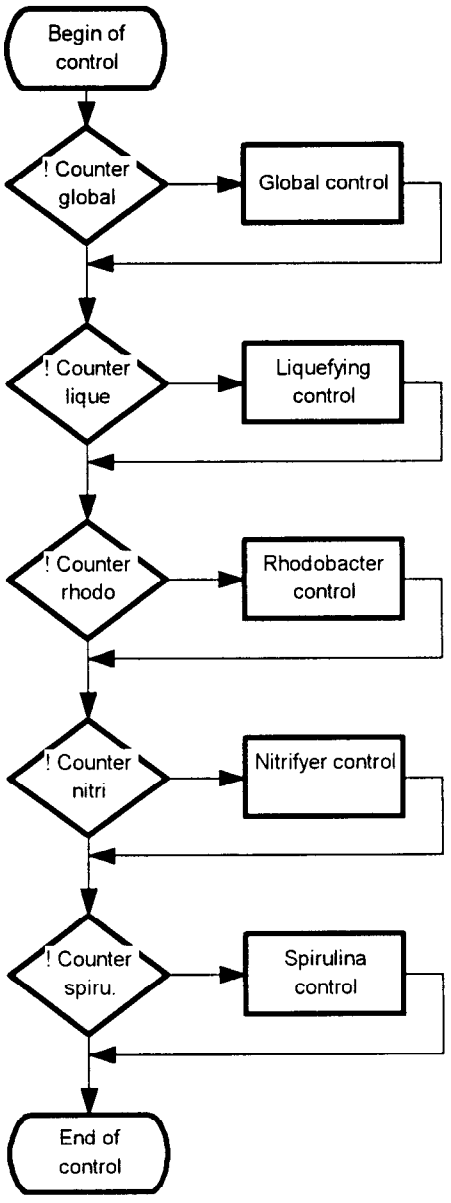


Figure 3

[2] A bitmap is a number, a number of integer type, but it is used as a set of switches, being a switch a binary digit. It is usually aplied to send a group of boolean values. For further details refer to source code.

1.8.5 .- Disk storage.

Although the disk storage done by the other stations is not changed, the disk storage in the GPS has been conceived to give the possibility to store the main managed variables. The variables to store were selected by the development team and are all the variables that can be selected to plot.[3] Due to the system does not have a high number of variables that could be interesting to store, the developers decided to store hardly everything.

This module saves to disk the data specified as input. This input is provided by the process control module.

[3] See the table in Addendum A, point 2.

1.9 .- Check the keyboard.

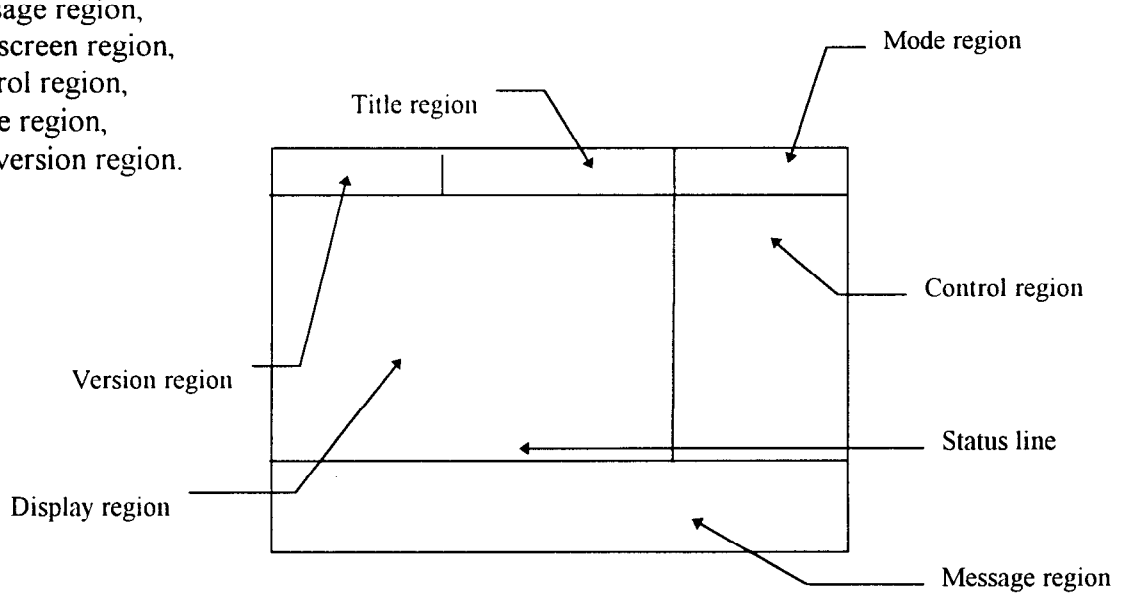
A function must be provided to allow the user to select one screen from the group of screens that now are available using the keyboard. This module is in charge of this task, scans the keyboard for an input, in case of a keyboard event, it takes appropriate action by changing the actual screen display or call other routines if necessary.

2.- User interface.

2.0.- Screen display description.

After initialisation the screen is divided into six regions :

- display region,
- message region,
- title screen region,
- control region,
- mode region,
- and version region.



The data that appear in each region are organised as follows, ...

2.0.1.- Display region.

This is the most important region, its task is to show the compartment information, display a graphical data representation, or any kind of menus and options. To maintain the consistence with the actual state, it is updated every minute.

The last line of this rectangle is booked for messages informing about the system status. To allow an easier way to manage this part of the screen, a text window has been defined.

2.0.2.- Message region.

This is the bottom window, its purpose is to show all the messages generated by the GPS. This messages report on what operation is being executed and any error or warning status.

2.0.3.- Title screen region.

It is located in the middle top of the screen. It is used to describe the type of screen being displayed.

2.0.4.- Control region.

It is located on the right side of the screen. Its purpose is to give the user information independently of the information visualised on the display region. It shows the date and time, the group that is currently operating, and the minutes left for a control in each reactor - every minute, or every configured interval, a sample is taken, and it must not be confused with a control period -. It is also planned, for a later version, to include all the alarms pilots in this region too.

2.0.5 .- Mode region.

Sometimes, for maintenance or developing tasks, it may be desirable to have another GPS program running in another computer, but only to follow the process but not to control. This takes place for example when testing a new version of the GPS on which only the screens have been modified. In this sense, two GPS have been made, one for control and another for monitorize the system.

The difference between them consists on conceiving the first as a complete GPS, and the second a GPS without the possibility to write on the net. In this way it is possible to have one control station and several monitoring station working together.

This region gives the information on which one is the type of the GPS running, the monitoring station or the controller station.

2.0.6.- Version region.

This is the window located at the top and left of the screen, it shows the current version of the GPS software , in this case 2.1

3.- Program controls.

3.1.- How to end a session.

The way to exit from the program is the same that in the previous version, It consist in pressing Control-C and ratifying. The corresponding flow chart can be seen in Figure 4.

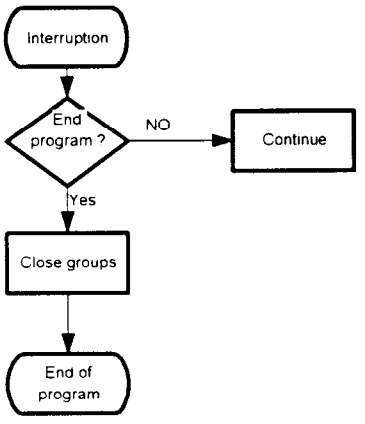


Figure 4

3.2.- Help system.

Although the system is not difficult to use, if necessary a help screen can be displayed. In the present case it shows the functions of the different keys available

3.3.- Text compartment information.

The presentation of the data has been divided in two different screens, a graphical one and a text one. The text screen gives the current values of the variables with the required precision. In the *Spirulina* compartment redistribution of the variables displayed has been done for easier identification and to allow more data to be displayed.

As can be seen, basically there are four text screen compartments :

- Spiruline compartment.
Fully operative and ready to use.
- Nitrifying compartment.
Test data being obtained is displayed.
- Rhodobacter compartment.
Ready for connection.
- Liquefying compartment.
Ready for connection.

3.4.- System conditions (F2)

The GPS is a program that must be working continuously. In this case it would be desirable to be able to check the values of the activated parameters being used. This screen has been included for this purpose.

The information shown at this time is the alpha coefficient of the filters, the sampling period of each compartment, if the alarms are set or not, and the disk status.

The point about the alarms has been made in order to allow the user to suppress the alarm sound.

The disk status tag is just a message that warns up that the hard disk is becoming full, or there is no problem to keep on working.

3.5.- File options (F3)

In case of a full hard disk status, it is necessary to be able to generate free space without stopping the GPS controller task. The GPS now includes a mechanism to manage the disk storage. There are some restrictions to this facility. There can only be moved, removed or copied files that are the output of the GPS, that is to say, log files and output files, but never executables - It cannot be accepted that the user deletes the GPS program or a configuration file - or any other kind of files.

3.5.0.- Information storage.

The previous GPS program version relied on the file storage facilities that were implemented in the software of the system station. This kind of storage has been maintained. However a parallel disk storage system has been foreseen for this station. In this version, a simple file storage routine has been implemented. It allows to obtain one data file per day with

the main GPS variables stored in ASCII code. This method allows to obtain, in a simple way, data in a highly transportable format. The only limit to this is the available disk space. When the disk space is exhausted, the user is prompted to remove the files from the disk and the storage is disconnected. This way the old data is maintained. This method is complementary to the one taking place at the system station where the old data is overwritten.

The naming convention of the files is :

ccyymmdd.out

where "cc" stands for compartment :

sp - spiruline
nt - nitrobacter
lq - liquefying
rh - rhodobacter

"yy" for year, "mm" for month, and "dd" for day. As an example the file named sp960621.out should be the spiruline compartment file generated from the outputs of the 21st of June of 1996.

3.5.1.- Copying information files.

This option, asks for a file name, and makes a copy of the file to the drive A:. It tests for the existence of a disk in drive A ; and the free space needed in it to avoid a system error.

3.5.2.- Moving information files.

This option allows the removal of a file from the hard disk to a floppy disk. Its operation is analogous to the previous one.

3.5.3.- Copying the log file to a floppy.

This function allows to copy of the log to a floppy. It allows for its revision in another computer.

3.5.4.- Deletion of a log file.

This functions allows the deletion of a log file in order to free disk space.

3.5.5.- Directory list.

This option allows to list all the removable files from the disk. This corresponds to the files with ".out" and ".txt". It also displays the disk space available on the disk.

3.6.- Graphical representations.

This function has been added in order to be able to have a graphical representation of the data. The program asks for the starting and ending dates of the period during which to display the variables of the configuration file

When a graphical display is on, by pressing the keys "+" and "-", the vertical scale changes between all the variables that are in plot, identifying them by their colour. Also, the y axis can be rescaled pressing the keys '*' or '/'.

3.6.1.- Expansion of the system.

Taking into account that this is a system in development and it will be necessary to expand and improve it in the near future, this software has to be designed to allow an easy expansion as soon as new compartments are developed or the existing ones are improved. In the current implementation this has been one of the design specifications. To achieve this goal for the graphical display of the data, the actual data to be displayed has been made independent of the plotting functions. The name of the variables to be displayed is read from the configuration files and included in the "datagraph" structure. It exist one of this structures for each compartment to be displayed. The actual names of variables to be displayed is accessed through a pointer. The pointed structure to be displayed changes according to the keyboard input indicating the compartment data to be displayed (Figure 5).

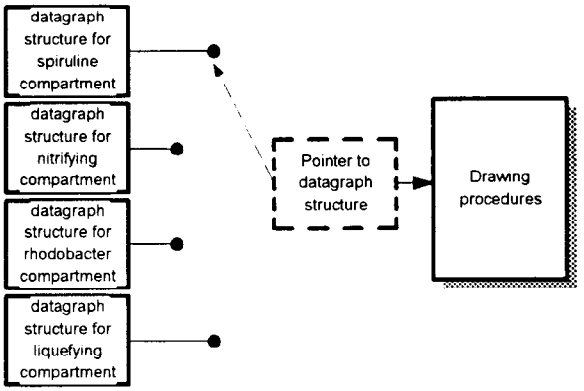


Figure 5

4 .- Signal treatment.

A potential problem to avoid is the noise on the measured data, this noise is generated by the lack of precision of some measurement instruments, and has the property of being a high frequency signal. As required by TN 28.2 (4.4.3) a high frequency filter has been implemented.

$$s(t+1) = s(t) * \alpha + i(t+1) * (1-\alpha)$$

Where s(x) is the filtered value at time x, and i(x) is the real measured input in time x. The important point is to get a good alpha coefficient. By default, no filters are active that is to say for each input that can have noise, its alpha is zero, but all the coefficients can be changed in the configuration files as explained in addendum 1.

To give the chance of the inclusion of a more sophisticated filter function, the following functions have been developed to make that work easier.

```
void filter(VARS *var)
{
float alpha;
float realval;

alpha = selectcoefficient (var->name);

realval = var->val[var->i];     /* Measured value */
var->val[var->i] = alpha * var->val[ var->i - 1 % (NB_SAMP + 1)] +
      (1-alpha) * realval;
var->value = var->val[var->i];

}


float selectcoefficient (char *name)
{
if (!strcmp (name, "LOOP0107")) return ALPHAFILTERcxa;
if (!strcmp (name, "LOOP0103")) return ALPHAFILTERnitrate;
if (!strcmp (name, "LOOP0105")) return ALPHAFILTEREb;
if (!strcmp (name, "LLOP0104")) return ALPHAFILTERpH;

return 1;
}
```

Then, to change the filters, the programmer only will have to change this two simple functions by others.

Addendum A .- Configuring the GPS.

A.0 .- Introduction

The GPS has a group of conditions that are subjected to change, these are the sampling time (in minutes) , filter's coefficients, variables and colours to plot. There are other aspects that could be subjected to change, but this ones are proposed here as examples, and could be expanded in future modifications.

The configuration is divided in two groups of files, the ones that concern to the graphical results, and the ones that concern to the control of the reactor.

A.1 .- Configuration of the control.

The file read by the GPS to take its configuration is the **melissa.cfg**, and the syntax used is shown in Figure 6.

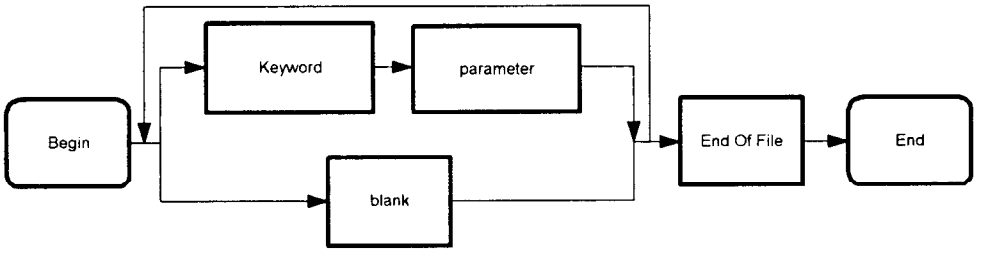


Figure 6[4]

In this first version, the *keywords* are the following :

Keyword	Parameter	Concerns to ...
FILTERCXA	Real value between 0 and 1	Alpha coefficient applied to the cxa variable filter call.
FILTERNITRATE	Real value between 0 and 1	Alpha coefficient applied to the nitrate filter call.
FILTEREB	Real value between 0 and 1	Alpha coefficient applied to the Eb variable filter call.
FILTERPH	Real value between 0 and 1	Alpha coefficient applied to the Ph variable filter call.
SAMPSPIRU	Integer value	Interval between samples for the spiruline compartment.
SAMPRHODO	Integer value	Interval between samples for the rhodobacter compartment.
SAMPLIQUE	Integer value	Interval between samples for the liquefying compartment.
SAMPNITRI	Integer value	Interval between samples for the nitrifying compartment.

[4] This diagram shows the regular grammar used to configure the GPS, it means that all keywords must be followed by a parameter, and after a parameter it can be found a keyword or an end of file. If that rules are not respected and the input file cannot be represented with this grammar, the configuration file will be wrong.

This means that to atribute a filter to a variable, or to set a sample interval, the configuration file should have the corresponding keyword to the variable and the value of the filter, or the sample time. A filter can only be used with the variables that are the result of a measure.

An example of a configuration file, used for testing, is ...

filternitrate 0.3
filtercxa 0.3
filterph 0.3
filtereb 0.3

samprhodo 2
samplique 4

A.2 .- Configuring the GPS graphical capabilities.

Other files are used to configure the graphical representation offered when ALT+F5 is pressed. These files are : ***spgraph.cfg, lqgraph.cfg, ntgraph.cfg, rhgraph.cfg***. At the present time, only the first is used, but everything is planed to make an easy expansion.

This files have a different look than the one explained before, that is to say, the first could have a random length, this one does not. This has a fixed length, and there is no grammar to represent it. It consist in **nine** lines, with four fields each line. The first field means if the variable is going to be plotted or not - a zero means no, and a one means yes -, the second number is the colour assigned to the variable, if it is not plotted the number must be there anyway. The third and fourth numbers indicates the scale wished for representation, the program will insert marks in the vertical axis in the middle and quarters of the interval, so if the interval is [0,4], marks are going to appear in (0, 1, 2 , 3, 4), but if the interval is [0,5] those marks are going to be now in (0, 1.25, 2.5, 3.75, 5), so the scale is predictable. In any case, a new feature included is the chance of changing the scale in run time, the vertical scale can be multiplied , or divided, by two as many times as the user could need.

The fifth element of the line, is a string, whose maximum length is **fourteen** characters, this string is the name that is going to appear in the legend of the graphical representation, and it is going to appear in the colour specified in the first number.

Each line is associated with one variable, strictly, the first line is associated with the first number in the output files, the second with the second and so on. As it has been configured when writing this lines, the relations are represented in the next table :

Number of the variable	Internal variable used.
1	Average of the variable cxa during the last 10 minutes.
2	Average of the variable nitrate during the last 10 minutes.
3	biomass production
4	temperature
5	cons_prod_real.sp
6	qe_real.sp
7	qe_real.value
8	Fr.sp
9	Eb.sp

In the case of any change, please, note that in all *.out files, the first line tells the variables saved there.

Addendum B .- Sources and comments.

In this addendum it can be found all the functions used in the GPS with a little comment and an uniform format to make easy the consultation. The functions are separated by modules, and every module is associated to a file with ".c" extension.

B.1 .- Module : Melissa

Module : Melissa.c **Name :** remove_file

Parameters : char *name

Output : none

Description : It asks the user to ratify the decision, and it removes the file named name. In case of being impossible to do it, it advises the user.

Source code :

```
void remove_file (char *name)
{
char c;

display_status ("Are you sure ? (Y/N)");
printf ("\a\a");

while (!kbhit());
c = getch();
if (c!= 'y' && c!= 'Y')
                {
                display_status(" ");
                return;
                }


if (remove(name)==-1)          /* If error removing file ... */
        {
        sprintf (buffer,"Could not delete the file %s",name);
        display_status(buffer);
        printf("\a\a");
        display_status (" ");
        wait_time(2);
        }
display_status(" ");
}
```

Module : Melissa.c **Name :** move_files

Parameters : char *source, char *dest

Output : none

Description : It moves the file called source to a destination called dest. The process is check the floppy, call the copy function, and if no error has occurred remove the source. It has been done to copy always to a floppy drive.

Source code :

```
void move_files (char *name, char *destination)
{
unsigned long lliure;

/* Verify the floppy drive and free space */
if (strcmp (destination, "NUL") && strcmp(destination, "nul") )
    {
    lliure = checkfloppy();                    /* Check if file is ok */
    if (lliure == -1) {
                      display_status ("Drive A: is not ready.");
                      printf ("\a\a");
                      wait_time(2);
                      display_status (" ");
                      return;
                      }
    }

if (copia(name, destination, lliure)==0)       /* Copy the file */
        {   /* If ok => remove the original */
        if (remove(name)==-1)
                {
                sprintf (buffer,"Could not delete the file %s",name);
                display_status(buffer);
                printf("\a\a");
                display_status (" ");
                wait_time(2);
                }

        }

}
```

Module : Melissa.c **Name :** copia

Parameters : char *source, char *dest, unsigned long frspace

Output : int, <0 if error occured, 0 if other case.

Description : This file, given a source file name, a destination file name and the amount of free space, if it feeds, copies the source file to a destination file called "dest".

Source code :

```
int copia (char *or, char *des, unsigned long lliure)
{
FILE *o, *d;
char c;

o = fopen (or,"rb");
if (o==NULL)  {
              display_status ("Error : source file does not exist.");
              printf("\a\a");
              wait_time(2);
              display_status (" ");
              return -1;
              }
fseek(o, 0L, SEEK_END);
if (lliure <= ftell(o)) {
                        display_status ("Not enough space in drive A:");
                        printf ("\a\a");
                        wait_time(2);
                        display_status (" ");
                        return -5;
                        }
fseek (o,0L,SEEK_SET);

d = fopen (des, "wb");
if (d==NULL)  {
              display_status ("Error opening destination file.");
              printf("\a\a");
              wait_time(2);
              display_status (" ");
              return -2;
              }

fread(&c, 1,sizeof(char), o);
do  {                                              /* perform the copy */
    fwrite(&c, 1, sizeof(char), d);
    fread (&c, 1, sizeof(char), o);
    } while (!feof(o) && !ferror(o) && !ferror(d) );

if (ferror(o)) {
               display_status ("Error during the copy process in source file.");
               printf("\a\a");
               wait_time(2);
```

```
                display_status (" ");
                return -3;
                }
if (ferror(d)) {
               display_status ("Error during the copy process in destination
file.");
               printf("\a\a");
               wait_time(2);
               display_status (" ");
               return -4;
               }

return 0;
}
```

Module : Melissa.c **Name :** checkfloppy

Parameters : none

Output : int. -1 if error, the amount of free space in the

Description : This function test the floppy drive, it is used to avoid a operating system error in the case that the floppy drive is empty and the program is trying to write to the floppy unit.

Source code :

```
long checkfloppy (void)
{
struct diskinfo_t diskinfo;
struct diskfree_t diskfree;
int st;
unsigned long lliure;
/*char bf[512];*/

diskinfo.drive = 0; /* ud A: */
diskinfo.head = 0;
diskinfo.track = 1;
diskinfo.sector = 2;
diskinfo.nsectors = 1;

st = _bios_disk (_DISK_VERIFY, &diskinfo);            /* Call to bios function */
if (0x8000 & st) return -1;

_dos_getdiskfree(1, &diskfree);

lliure = (unsigned long) diskfree.avail_clusters *
         (unsigned long)  diskfree.sectors_per_cluster *
         (unsigned long) diskfree.bytes_per_sector;


return lliure;
}
```

Module : Melissa.c **Name :** check_disk

Parameters : none

Output : none

Description : This is the function that has the mission of testing the hard disk for free space. It also sets the warning state or danger state variables to warn about the amount of free space.

Source code :

```
void check_disk (void)
{
int disk;
struct diskfree_t df;
unsigned long lliure;
char buffer[100];

display_status ("Checking disk ... ");

_dos_getdiskfree(0, &df);                          /* Default drive. */
                                    /* Calculate the amount of free space */

lliure = (unsigned long) df.avail_clusters * (unsigned long)
      df.sectors_per_cluster * (unsigned long) df.bytes_per_sector;

if (lliure < 100000) {diskfull=1; }


if (lliure < 400000) {
                     /* Warning */
                     display_error ("Disk nearly full !!!");
                     diskwarning = 1;
                     }
                     else
                     { diskwarning = 0;
                       diskfull = 0;
                     }
if (lliure < 100000) {
                    /* Disable disk saving */
                    sprintf (buffer,"Disabling disk saving ...");
                    display_error (buffer);
                    return;
                    }
return;
}
```

Module : Melissa.c **Name :** set_defaults

Parameters : none

Output : none

Description : It sets all the configurable variables to a default values. It is used by the paramenters_acquisition function.

Source code :

```
void set_defaults(void)
{
ALPHAFILTERcxa = 1;
ALPHAFILTERnitrate = 1;
ALPHAFILTEREb = 1;
ALPHAFILTERpH = 1;
SAMP_SPIRU = 1;
SAMP_LIQUE = 1;
SAMP_RHODO = 1;
SAMP_NITRI = 1;
}
```

Module : Melissa.c **Name :** parameters_acquisition

Parameters : none

Output : none

Description : Its task is to read the configuration file, and apply the contents of that file. In the case that an error occurs - or that file does not exist -, a default values are taken.

Source code :

```
void parameters_acquisition(void)
{
FILE *fp;
char input[100];
int ok=0;
float inputval=0;
long inputlong;

set_defaults();
fp=fopen(CONFIG_FILE, "rt");
if (fp==NULL) { /* Set defaults values */
            tech_log_file (" Warning ... no configuration file.");
            return;
            }

while (!feof (fp))
      {
      ok=0;
      fscanf (fp, "%99s", input);
      if (strcmp(input, ""))
            {
            char *uperinput;
            uperinput = strupr(input);

                        /* Chech for a keyword and the corresponding parameter */
            if (!strcmp (uperinput, "FILTERCXA")) {
                  fscanf (fp, "%f", &inputval);
                  ALPHAFILTERcxa = inputval;
                  ok++;
                  }
            if (!strcmp (uperinput, "FILTERNITRATE")) {
                  fscanf (fp, "%f", &inputval);
                  ALPHAFILTERnitrate = inputval;
                  ok++;
                  }
            if (!strcmp (uperinput, "FILTEREB")) {
                  fscanf (fp, "%f", &inputval);
                  ALPHAFILTEREb = inputval;
                  ok++;
                  }
            if (!strcmp (uperinput, "FILTERPH")) {
```

```
                fscanf (fp, "%f", &inputval);
                ALPHAFILTERpH = inputval;
                ok++;
                }
        if (!strcmp (uperinput, "SAMPSPIRU")) {
                fscanf (fp, "%i", &inputlong);
            SAMP_SPIRU = inputlong;
                ok++;
                }
        if (!strcmp (uperinput, "SAMPRHODO")) {
                fscanf (fp, "%i", &inputlong);
            SAMP_RHODO= inputlong;
                ok++;
                }
        if (!strcmp (uperinput, "SAMPLIQUE")) {
                fscanf (fp, "%i", &inputlong);
            SAMP_LIQUE = inputlong;
                ok++;
                }
        if (!strcmp (uperinput, "SAMPNITRI")) {
                fscanf (fp, "%i", &inputlong);
            SAMP_NITRI = inputlong;
                ok++;
                }
            if (!ok) { /* Error reading parameter file, keyword not valid */
                                /* Set default values */
                char buffer[200];
                set_defaults();
                /* Error message */
                sprintf (buffer, "Unknown parameter : %s",input);
                display_error(buffer);
                _clearscreen(_GWINDOW);
                }
            }
      }

}
```

Module : Melissa.c **Name :** disk_storage

Parameters : int input

Output : none

Description : It stores the information related to the compartments indicated in the input bitmap.

Source code :

```
void disk_storage (int input)
{
FILE *fp;
double v1, v2;
char buffer[400], file[20];
time_t ltime;
struct  tm  *dt;

if (diskfull) return;

if (input & 0x01)       /* Spirulina output */
      {
    strcpy (file,"sp");
    time(&ltime);
    dt=localtime(&ltime);
    dt->tm_mon++;
    sprintf(file,"sp%02d%02d%02d.out",dt->tm_year,dt->tm_mon,dt->tm_mday);

    fp = fopen (file, "rt");
    if (fp==NULL) {           /* Cannot open file -> new file  */
                  char buffer[400];

                  fp = fopen (file, "at");          /* Set header */
                  sprintf (buffer, "cxa_moy, nit_moy, production.sp,
react.temp, cons_prod_real.sp,  qe_real.sp, qe_real.value, Fr.sp, Eb.sp\n");
                  fwrite(buffer, 1, strlen(buffer), fp);

                 }
        else    /* if file can be opened -> go to end of file */
            {
            fclose (fp);
            fp = fopen (file, "at");
            }

    if (fp==NULL) { display_error ("Unable to open output file");
                  return; }
    v1 = average_var(&cxa, 10);
    v2 = average_var(&nitrate, 10);
    sprintf (buffer, "%02d:%02d %f %f %f %f %f %f %f %f\n",dt->tm_hour,dt-
>tm_min,
        v1 , v2 , production.sp, temperature.value,
        cons_prod_real.sp,  qe_real.sp, qe_real.value, Fr.sp, Eb.sp);
```

```
        fwrite (buffer, 1, strlen(buffer), fp);
        fclose (fp);
        }

}
```

Module : Melissa.c **Name :** alarms

Parameters : none

Output : none

Description : Its task is to call all the alarm functions for all the compartments.

Source code :

```
void alarms (void)
{
char buffer[100];

alarm_spiru();
alarm_lique();
alarm_nitri();
alarm_rhodo();

sprintf (buffer, "Acquisition of alarms.\n");
_outtext (buffer);
wait_time (1);

_clearscreen(_GWINDOW);
}
```

Module : Melissa.c **Name :** control_process

Parameters : ScreenViews active_reactor

Output : int

Description : This function calls the control functions for each compartment if the associated counter has reached zero. The output is a bitmap that indicates which compartment control has been running and must store information in the disk.

Source code :

```
int control_process (ScreenViews active_reactor)
{
int out=0;
char buffer[100];


if ((--min_glob)==0)
    {
            /* Should call to global control */
    min_glob = SAMP_GLOB;
    }

if((--min_lique)==0)
      {
      if (control_lique(active_reactor)) out = out | 0x02;
    min_lique = SAMP_LIQUE;
      }

if((--min_rhodo)==0)
    {
    if (control_rhodo(active_reactor)) out = out | 0x08;
    min_rhodo = SAMP_RHODO;
    }

if((--min_nitri)==0)
      {
      if (control_nitrogen(active_reactor)) out = out | 0x04;
    min_nitri = SAMP_NITRI;
      }

if((--min_spir)==0)
    {
    if (control_spiru(active_reactor)) out = out | 0x01;
    min_spir = SAMP_SPIRU;
    }

_clearscreen(_GWINDOW);

if (out & 0x01 && active_reactor == gspirulina) result (&active_reactor,0);
if (active_reactor != gspirulina && active_reactor != igspirulina)
    result (&active_reactor,0);
```

```
    #ifndef __Monitoring
    else
            next_control ();
    #endif

return out;
}
```

Module : Melissa.c **Name :** keybdrv

Parameters : ScreenViews* react, char inp

Output : none

Description : This is the routine that control the keyboard inputs, given the actual screen (react) and an input character, it takes the proper decision.

Source code :

```
void keybdrv (ScreenViews *react, char chr)
{
int modified=0;
int new=1;

if (!chr)   /* Function keys */
    {
    chr = getch();
    switch (*react)
    {
    case filemanagement:          /* File managemant submenu. */
        {
        if (chr== 0x3b) { *react= help_1;           /* F1 */
                  ++modified;
                }
        if (chr== 0x3f) { move_disk();                /* F5 */
                          ++modified;
                }
        if (chr== 0x40) { move_files ("log.txt", "A:log.txt"); /* F6 */
                }
        if (chr== 0x41) { remove_file ("log.txt");      /* F7 */
                        }
        if (chr== 0x42) { directory ();      /* F8 */
                         *react = principal;
                        }
        if (chr== 0x43) { copy_disk();                /* F9 */
                          ++modified;
                }

        } break;
    default:            /* Main menu */
        {
        if (chr== 0x3b) { *react= help_1;           /* F1 */
                  ++modified;
                }
        if (chr== 0x3c) { *react= status;           /* F2 */
                  ++modified;
                }
        if (chr== 0x3d) { *react= filemanagement;           /* F3 */
                  ++modified;
```

```
            }
    if (chr== 0x3f) { *react = spirulina;        /* F5 */
              ++modified;
            }
    if (chr== 0x6c) { *react = gspirulina;        /* Alt+F5 */
                      ++modified;
                    }
    if (chr== 0x62) { *react = igspirulina;       /* Crt+F5 */
                      ++modified;
                    }
    if (chr== 0x40) { *react = reactor_2;        /* F6 */
              ++modified;
            }
    if (chr== 0x41) { *react = reactor_3;        /* F7 */
              ++modified;
            }
    if (chr== 0x42) { *react = reactor_4;        /* F8 */
              ++modified;
            }
    if (chr== 0x68) {                              /* ALT+F1 */
          screen_init();
          ++modified;
          }
    }
    break;
  }   /* End of switch */
} /* End of if keypressed = F* */
else
{
if (chr == 27) { *react = principal;              /* Escape */
            ++modified;
          }
       else
       {
       switch (*react)
            {
            case gspirulina :
            case igspirulina:
                    {
                    switch (chr)
                        {
                        case '+' : AdvanceScale ();
                                   break;
                        case '-' : GoBackScale();
                                   break;
                        case '*' : IncrementScale ();
                                   new=0;
                                   ++modified;
                                   break;
                        case '/' : DecrementScale();
                                   new=0;
                                   ++modified;
                                   break;
                        default : break;
                        }
                    break;
                    }
```

```
                }
            }          /* End of else */
    }
if (modified) { if (!result(react,new)) {result(react, new); }}
}
```

Module : Melissa.c **Name :** my_interrupt

Parameters : none

Output : none

Description : It is the function called when control-c is pressed. It asks the user to ratify the end of the execution

Source code :

```
int     my_interrupt()
      {
      char    ch;
      signal(SIGINT,SIG_IGN);
      _clearscreen (_GWINDOW);
      _outtext("Terminate processing ? ");
      ch=getch();
      if((ch=='y')||(ch=='Y'))
            {
            close_grp_gps();
            _outtext("\nbye.... *** Program Terminated by user ***\n");
            wait_time(5);
#ifndef ADERSA
            _setvideomode(_DEFAULTMODE);
#endif
            exit(0);
            }
      if( signal(SIGINT,my_interrupt) == (int(*)())-1)
            {
            _outtext("\nCouldn't set SIGINT *** Program Terminated ***\n");
            exit(0);
            }
      _outtext("Continue...\n");
      return;
      }
```

B.2 - Module : Alarms

Module : alarms.c **Name :** alarm_rhodo

Parameters : none

Output : none

Description : In the future, in this function will be placed the source code for the alarms for the rhodobacter compartment.

Source code :

```
void alarm_rhodo(void)
{
}
```

Module : alarms.c **Name :** alarm_nitri

Parameters : none

Output : none

Description : In the future, in this function will be placed the source code for the alarms for the nitrifying compartment.

Source code :

```
void alarm_nitri(void)
{
}
```

Module : alarms.c **Name :** alarm_lique

Parameters : none

Output : none

Description : In the future, in this function will be placed the source code for the alarms for the liquefying compartment.

Source code :

```
void alarm_lique(void)
{
}
```

Module : alarms.c **Name :** send_alarm

Parameters : none

Output : none

Description : This function resets all the alarms, this avoid the operator to go to the MICON and press manualy the button.

Source code :

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

      write_var(&rst_alarm1);
      write_var(&rst_alarm2);
      write_var(&rst_alarm3);
      write_var(&rst_alarm4);
      write_var( &rst_main_alarm );
      write_var( &rst_micon1_alarm );
      write_var( &rst_pressure);

      wait_time(2);
      display_status(" ");
      }
```

Module : alarms.c **Name :** acq_alarm

Parameters : none

Output : none

Description : Its task is to read the value of the variables that represent alarms.

Source code :

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

      read_var(&rst_alarm1);
      read_var(&rst_alarm2);
      read_var(&rst_alarm3);
      read_var(&rst_alarm4);
      read_var( &rst_main_alarm );
      read_var( &rst_micon1_alarm );
      read_var( &rst_pressure);
      read_var( &alarm_micon1 );
      read_var( &alarm_micon1_value);


      wait_time(2);
      display_status(" ");
      }
```

Module : alarms.c **Name :** init_alarm

Parameters : none

Output : none

Description : This function sets the names of those variables that represent alarms.

Source code :

```
init_alarm()
      {
      display_status("Initialisation of ALARMS ...");

      sprintf (rst_alarm1.name, "VD--0141");
      sprintf (rst_alarm2.name, "VD--0145");
      sprintf (rst_alarm3.name, "VD--0149");
      sprintf (rst_alarm4.name, "VD--0153");
      sprintf (rst_main_alarm.name, "VD--0155");
      sprintf (rst_micon1_alarm.name, "VD--0156");
      sprintf (rst_pressure.name, "VD--0144");
      /*sprintf (active_lamp.name, "VD--0146");*/
      sprintf (alarm_micon1.name, "LOC-0122");
      sprintf (alarm_micon1_value.name, "LOC-0121");

      wait_time(2);
      display_status(" ");
      }
```

Module : alarms.c **Name :** alarm_spiru

Parameters : none

Output : none

Description : This is the function that checks the spiruline compartment for irregularities. If irregularities are found, send_alarm function is called.

Source code :

```
void alarm_spiru()
      {
      long alval;
      int al=0;
      char bf[100];

      if (!activated_alarms) return;  /* To give the chance to disconnect
                                         all the alarms.              */
      acq_alarm();
      if (alarm_micon1.value != 0 )
        {
        al++;
        alval = (long) alarm_micon1_value.value;
        if (alval & 0x01)
            {
                  /* Alarm on P100 1 */
                  display_error ("Alarm in Micon 1");
                        rst.alarm1.value = 1;
            }
        if (alval & 0x02)
            {
                  /* Alarm on P100 2 */
                  display_error ("Alarm in Micon 2");
                  rst.alarm2.value = 1;
            }
        if (alval & 0x04)
            {
                  /* Alarm on P100 3 */
                  display_error ("Alarm in Micon 3");
                  rst.alarm3.value = 1;
            }
        if (alval & 0x08)
            {
                  /* Alarm on P100 4 */
                  display_error ("Alarm in Micon 4");
                  rst.alarm4.value = 1;
            }
        }

        if (Fr.value < (Fr.sp * 0.9) || Fr.value > (Fr.sp * 1.1))
            {
```

```
                /* Light is not between limits */
                display_error ("Light is not between limits");
                al++;
        }

if (al) send_alarm();
}
```

B.3 .- Module : Vars

Module : vars.c **Name :** selectcoeffient

Parameters : char *name

Output : float

Description : Given a variable name, this function returns the alpha coefficient associated to this vairable name. if the name is not in the list, 1 is returned.

Source code :

```
float selectcoefficient (char *name)
{
if (!strcmp (name, "LOOP0107")) return ALPHAFILTERcxa;
if (!strcmp (name, "LOOP0103")) return ALPHAFILTERnitrate;
if (!strcmp (name, "LOOP0105")) return ALPHAFILTEREb;
if (!strcmp (name, "LLOP0104")) return ALPHAFILTERpH;

return 1;
}
```

Module : vars.c **Name :** filter

Parameters : VARS *var

Output : none

Description : This function filters the input value of the variable pointed by "var". This is a first order filter.

Source code :

```
void filter(VARS *var)
{
float alpha;
float realval;

alpha = selectcoefficient (var->name);          /* Get the coefficient */

realval = var->val[var->i];                     /* Mesured value */
var->val[var->i] = alpha * var->val[ var->i - 1 % (NB_SAMP + 1)] +
         (1-alpha) * realval;
var->value = var->val[var->i];

}
```

Module : vars.c **Name :** check_network

Parameters : none

Output : none

Description : This function task is to check the network, and if it does not work send a message to the log file.

Source code :

```
check_network()
      {
      int counter=0;  /* If the net is not working ... we generate a
                      infinit loop */
    char buffer[300];
      int rt;


      display_status("Checking Network ...");
      while((rt=garde(101,1)) && (counter++ < MAX_CHECK_NETWORK)) ;
    if (counter >= MAX_CHECK_NETWORK) log_file ("Network test not passed!");
      display_status(" ");
      }
```

Module : vars.c **Name :** error_gps

Parameters : none

Output : none

Description : This function, with the global variable gpserror, sends a error message to the function "display_error"

Source code :

```
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:ignored ...");
               /* 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;
                }
        }
}
```

Module : vars.c **Name :** write_gps

Parameters : VARS *write_var

Output : int <0 if error ocurs, 0 in other case

Description : This function is the responsible at lower level of writing a variable.
It control all errors described in the manual provided by ADERSA.
This function interacts with the functions provided by ADERSA.

Source code :

```
#ifndef ADERSA
int write_gps(VARS *write_vars)
#else
int write_gps(write_vars)
VARS *write_vars;
#endif
      {
      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);
          }
}
```

Module : vars.c **Name :** write_var

Parameters : VARS *write_var

Output : none

Description : This routine, given a VARS structure pointer, tryes to write a va riable in all groups until finds the group it belongs. If an error ocurs, my_interrupt() is called

Source code :

```
#ifndef ADERSA
void write_var(VARS *write_var)
#else
write_var(write_var)
VARS *write_var;
#endif

      {
      int     file_rank,ret_code;
      char    buffer[80];

#ifndef __Monitoring        /* It gives us the chance to have monitoring stations
*/

      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 when
writing",write_var->name);
                        display_error(buffer);
                        my_interrupt();
                        break;
                        }
                  }
            }
#endif
      }
```

Module : vars.c **Name :** read_gps

Parameters : VARS *read_var

Output : int <0 if error, the read value in other case.

Description : This function is the responsible at lower level of reading a variable.
It control all errors described in the manual provided by ADERSA.
This function interacts with the functions provided by ADERSA.

Source code :

```
#ifndef ADERSA
double read_gps(VARS *read_vars)
#else
double read_gps(read_vars)
VARS *read_vars;
#endif
      {
      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);
}
```

Module : vars.c **Name :** read_var

Parameters : VARS *read_var

Output : none

Description : This routine, given a VARS structure pointer, tryes to read a variable in all groups until finds the group it belongs. If an error ocurs, my_interrupt() is called

Source code :

```
#ifndef ADERSA
void read_var(VARS *read_var)
#else
void read_var(read_var)
VARS *read_var;
#endif
      {
      double  value;
#ifndef ADERSA
      double  read_gps(VARS *);
#else
      double  read_gps();
#endif
      int     file_rank;
      char    buffer[80];
#ifndef _debugging_vars          /* When debugging do not use the net. */
      value=read_gps(read_var);
#else
      value = 1;
#endif
      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 when
reading",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;
      }
```

Module : vars.c **Name :** set_gps

Parameters : GPS_FILE *gp

Output : none

Description : Acquisition of the gps pointer.

Source code :

```
#ifndef ADERSA
void set_gps(GPS_FILE *gp)
#else
void set_gps(gp)
GPS_FILE *gp ;
#endif

      {
      gps=gp;
      }
```

Module : vars.c **Name :** init_vars

Parameters : none

Output : none

Description : Its task is to call the variable initialisation routines for each compartment.

Source code :

```
void init_vars (void)
{
    init_vars_spirulina();
    init_vars_nitrogen();
    /* More reactor variables to be add here */
}
```

B.4 .- Module : Screen

Module : Screen.c **Name :** set_title

Parameters : char *title

Output : none

Description : Given a string, this function centers it in the midle top of the screen.

Source code :

```
void set_title(char *tl)
      {
    int i;
    char bf[100];
    i = strlen(tl);
    if (i > 29) return;
    i = (int) i / 2;
    _settextwindow(2,25,2,55);
    _clearscreen(_GWINDOW);
    _settextposition(0, 15-i);
    _outtext (tl);
    use_message_window();
    }
```

Module : Screen.c **Name :** InputKey

Parameters : char *buffer, int maxlength, int onlynumbers

Output : int (0 if esc has been pressed, 1 in other case).

Description : Rutine for getting a string from the keyboard, the keys pressed are stored in the parameter buffer, and as much there will be insered "maxlength" characters. If the paramerter onlynumbers is set to 1, only numerical characters will be accepted.

Source code :

```
int InputKey (char* buf, int maxim, int onlynumbers)
{
char c;
int ndx=0;

c = getch();
while (c!=13 && c!=27 && ndx < maxim)
    {
    if(onlynumbers!=0)              /* process input key */
      {
      if ( (c >= '0' && c<='9')|| c==8 )
        {
        buf[ndx++]=c;
        printf ("%c",c);
        }
      }
      else
      {
      buf[ndx++]=c;
      printf ("%c",c);
      }
    if (c==8) {
              printf (" \b");
              ndx-=2;
              buf[ndx]='\0';
            }
    c=getch();                      /* Get another key */
    }
buf[ndx]='\0';
if (c==27) buf[0]=0;
if (!ndx || c==27) return 0;
return 1;
}
```

Module : Screen.c **Name :** get_input_string

Parameters : char *buffer, int x, int y, int maxlength

Output : int (0 if esc has been pressed, 1 in other case).

Description : This routine, places the cursor on (x,y) in the display region, and reads a string for as much "maxlength" characters.

Source code :

```
int get_input_string (char *buffer,short x,short y, int maxim)
      {
      struct  rccoord txtpos;
    int rt;

    if((x>52)||(y>20))
            {
            display_error("Can't display result : coordinates error on :");
            display_error(buffer);
            return(-1);
            }
      txtpos=_gettextposition();

      use_display_window();
      _settextposition(y,x);

      rt=InputKey(buffer, maxim ,0);

      use_message_window();
      _settextposition(txtpos.row,txtpos.col);
    return rt;
    }
```

Module : Screen.c **Name :** get_input_number

Parameters : char *buffer, int x, int y, int maxlength

Output : int (0 if esc has been pressed, 1 in other case).

Description : This routine, places the cursor on (x,y) in the display region, and reads a string for as much "maxlength" characters, but this character are only going to be numericals.

Source code :

```
int get_input_number (char *buffer , short x,short y, int max)
      {
    int rt;
    struct  rccoord txtpos;

    if((x>52)||(y>20))
            {
            display_error("Can't display result : coordinates error on :");
            display_error(buffer);
            return(-1);
            }
      txtpos=_gettextposition();

      use_display_window();
    _settextposition(y,x);

      rt=InputKey(buffer, max ,1);

      use_message_window();
      _settextposition(txtpos.row,txtpos.col);
    return rt;
    }
```

Module : Screen.c **Name :** display_activ_group

Parameters : GPS_FILE *gps

Output : none

Description : Given a pointer to the ring of files, it displays the active one.

Source code :

```
display_activ_group(GPS_FILE *gps)
      {
      char    buffer[100];
      struct  rccoord txtpos;
      txtpos=_gettextposition();

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

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

Module : Screen.c **Name :** display_no_activ_group

Parameters : none

Output : none

Description : Its task is to display that there is no group active.

Source code :

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

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

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

Module : Screen.c **Name :** display_time

Parameters : char *time

Output : none

Description : Given a string with the time and date, it prints it in the right place.

Source code :

```
display_time(char *buffer)
      {
      struct  rccoord txtpos;
      txtpos=_gettextposition();

      use_time_window();
      _outtext(buffer);

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

Module : Screen.c **Name :** display_result

Parameters : char *output, short x, short y

Output : none

Description : This function given a string and two coordinates, print the first where the second and third parameter indicate.

Source code :

```
display_result(char *buffer,short x,short y)
      {
      struct  rccoord txtpos;

    if((x>55)||(y>20))
            {
            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);
      }
```

Module : Screen.c **Name :** center_display_result

Parameters : char *output, short y

Output : none

Description : This function, centers in the y row, the string output. This is going to be done in the display region.

Source code :

```
center_display_result(char *buffer, short y)
      {
      struct  rccoord txtpos;
    int l;

    l = strlen(buffer);

    if(y>20 || l > 52)
            {
        display_error("Can't center display result : coordinates error on :");
            display_error(buffer);
            return(-1);
            }
      txtpos=_gettextposition();

    l = (int) l / 2;

      use_display_window();
    _settextposition(y,26-l);
      _outtext(buffer);

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

Module : Screen.c **Name :** get_input_string

Parameters : char *input, short x, short y

Output : none

Description : Given a pointer to a string, and two coordinates, this function waits for a keybord input in the display region. The introduced string is stored in "input".

Source code :

```
get_input_string (char *buffer,short x,short y)
      {
      struct  rccoord txtpos;

    if((x>52)||(y>20))
            {
            display_error("Can't display result : coordinates error on :");
            display_error(buffer);
            return(-1);
            }
      txtpos=_gettextposition();

      use_display_window();
      _settextposition(y,x);

    scanf ("%s", buffer);

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

Module : Screen.c **Name :** display_status

Parameters : char *string

Output : none

Description : It prints the input string to the status region.

Source code :

```
display_status(char *buffer)
      {
      struct  rccoord txtpos;
      txtpos=_gettextposition();

      use_status_window();
      _outtext(buffer);

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

Module : Screen.c **Name :** display_region

Parameters : char *error

Output : none

Description : This function prints the error in the message region, makes the bell ring, and call the output log file functions.

Source code :

```
display_error(char *buffer)
      {
        FILE *fp;


      _setbkcolor((long)GREEN);
      _settextcolor(RED+16);
      _outtext(buffer);
        printf("\a\a\a");
      _setbkcolor((long)BLUE);
      _settextcolor(WHITE);
      _wrapon(_GWRAPON);
      _displaycursor(_GCURSOROFF);
    error_log_file(buffer);
    _outtext("\n");

    }
```

Module : Screen.c **Name :** use_message_window

Parameters : none

Output : none

Description : It sets as active text window, the message region.

Source code :

```
use_message_window()
      {
      _settextwindow(26,2,29,78);
      _setbkcolor((long)BLUE);
      _settextcolor(WHITE);
      _wrapon(_GWRAPON);
      _displaycursor(_GCURSOROFF);
      }
```

Module : Screen.c **Name :** use_group_window

Parameters : none

Output : none

Description : It sets as active text window the group region.

Source code :

```
use_group_window()
      {
    _settextwindow(5,59,5,79);
      tab(3,4);
      _settextcolor(WHITE);
      _setbkcolor((long)BLUE);
      _clearscreen(_GWINDOW);
      }
```

Module : Screen.c **Name :** use_time_window

Parameters : none

Output : none

Description : It sets as active text window the time region.

Source code :

```
use_time_window()
      {
    _settextwindow(4,59,4,79);
      tab(58,4);
      _settextcolor(WHITE);
      _setbkcolor((long)BLUE);
      }
```

Module : Screen.c **Name :** screen_init

Parameters : none

Output : none

Description : Its task is to draw the main screen.

Source code :

```
void screen_init(void)
      {
    _setvideomode(_VRES16COLOR);
    _setbkcolor((long)RED);
    _settextwindow(1,1,25,80);
    _clearscreen(_GWINDOW);

    _settextcolor(BLACK);
    _settextcolor(5);

      _setbkcolor((long)BLUE);
      _settextcolor(WHITE);
#ifdef __Monitoring
    _settextposition(2, 59);
    _settextcolor(4);
    _outtext(" Monitoring station ");
#else
    _settextposition(2, 60);
    _settextcolor(4);
    _outtext(" Controler station ");
#endif
    _settextposition (2,2);
    _settextcolor(11);
    _outtext(" MELISSA GPS V2.00  ");

    _setcolor(1);
    _moveto(450, 0);
    _lineto (450,392);
    _moveto (640, 392);
    _lineto (0,392);
    _moveto(180, 39);
    _lineto (180,0);
    _moveto (0,0);
    _lineto (639,0);
    _lineto (639,473);
    _lineto (0, 473);
    _lineto (0,0);
    _moveto (639,39);
    _lineto (0,39);
    _setcolor (7);

    _settextwindow(4,2,24,55);
```

```
_clearscreen(_GWINDOW);

_settextwindow(26,2,29,78);
_clearscreen(_GWINDOW);
_settextwindow (4,58,24,79);
_clearscreen(_GWINDOW);

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

  }
```

Module : Screen.c **Name :** use_display_window

Parameters : none

Output : none

Description : It sets as active text window the display region.

Source code :

```
use_display_window()
      {
    _settextwindow(5,2,23,55);
    _settextcolor(WHITE);
      _setbkcolor((long)BLUE);
      tab(3,6);
      }
```

Module : Screen.c **Name :** InitializeGraphicalStructures

Parameters : none

Output : none

Description : This function initialises the function pointers used to acces the initilaising functions to be used during each redraw. and the title thas is going to appear in the screen.

Source code :

```
void InitializeGraphicalStructures (void)
{
spdata.initialisation = InitializeSpPlot;
strcpy (spdata.title, "Spirulina Compartment Graph");
}
```

Module : Screen.c **Name :** SelectSpirulineGraph

Parameters : none

Output : none

Description : This function, just sets the corresponding pointer to the spirulina dada structures.

Source code :

```
void SelectSpirulineGraph (void)
{
actdata = &spdata;
}
```

Module : Screen.c **Name :** IncrementScale

Parameters : none

Output : none

Description : This function, multiply by two the actual y axis.

Source code :

```
void IncrementScale (void)
{
actdata->max[lastscale] = actdata->max[lastscale] * 2;
}
```

Module : Screen.c **Name :** DecrementScale

Parameters : none

Output : none

Description : This function, divides by two the actual y axis.

Source code :

```
void DecrementScale (void)
{
actdata->max[lastscale] = actdata->max[lastscale] / 2;
}
```

Module : Screen.c **Name :** drawgraph

Parameters : int new, int interval

Output : int (0 if an error ocurred. 1 in any other case)

Description : This is the main function of graphical representation, the first parameter indicates if it has just been selected or now it is just repainting the screen. This function will ask the user for a beginning date and initiates the plot. The second parameter indicates if a representation between do dates is wished.

Source code :

```
int drawgraph (int new, int interv)
{
char cad[100], cad2[100];
long begin, end, nlines;
time_t ltime;
struct  tm  *dt;

use_display_window();
_clearscreen(_GWINDOW);
use_message_window();


if (new) {                      /* If new selection, ask for date or dates */
    actdata->initialisation();
    display_result ("Initial date : (yymmdd) ",5,8);
    use_display_window();
    _settextposition(8,30);
    if (!get_input_number(cad, 30, 8, 100)) return 0;
    if (interv)
        {
        display_result ("Ending date : (yymmdd) ",5,9);
        use_display_window();
        _settextposition(9,30);
        if (!get_input_number(cad2, 30,9, 100)) return 0;
        }
        else
        {
        time(&ltime);
        dt=localtime(&ltime);
        dt->tm_mon++;
        sprintf(cad2,"%02d%02d%02d",dt->tm_year,dt->tm_mon,dt->tm_mday);
        }
        use_message_window();
    }
    else          /* If not new, select the last begining date */
        {
        strcpy (cad, olddate);
        if (!interv)
```

```
            {
            time(&ltime);
            dt=localtime(&ltime);
            dt->tm_mon++;
            sprintf(cad2,"%02d%02d%02d",dt->tm_year,dt->tm_mon,dt->tm_mday);
            }
            else
            {
            strcpy (cad2, olddateend);
            }
        }


strcpy (olddate, cad);
strcpy (olddateend, cad2);

use_display_window();
_clearscreen(_GWINDOW);
use_message_window();

set_title (actdata->title);

begin = atol(olddate);
end = atol (olddateend);


nlines = calcule_between (begin, end);  /* Get the number of points to draw */
if ( nlines > 0)
        {
        DrawAxes (nlines, begin, end);
        if (Draw (nlines, begin, end)==-1)
            {
            _outtext ("Error during drawing process!!");
            }
        DrawLegend();
        if (new) AdvanceScale();
        DrawScale(lastscale);
        }
        else
        {
        display_status ("Unable to draw graph.");
        printf ("\a\a");
        wait_time(2);
        display_status (" ");
        use_message_window();
        return 0;
        }
return 1;

}
```

Module : Screen.c **Name :** tab

Parameters : short x, short y

Output : none

Description : It does exactly the same that the function _settextposition, is to say, it places the cursor in a wished position.

Source code :

```
tab(short x,short y)
      {
      _settextposition(y,x);
      }
```

Module : Screen.c **Name :** use_control_window

Parameters : none

Output : none

Description : It sets as active text window the control region.

Source code :

```
use_control_window()
      {
     _settextwindow(6,58,24,79);
     tab(3,18);
       _settextcolor(WHITE);
       _setbkcolor((long)BLUE);
       _clearscreen(_GWINDOW);
     }
```

Module : Screen.c **Name :** next_control

Parameters : none

Output : none

Description : Its task is to print in the control region the minutes left for the next process control for each compartment.

Source code :

```
void next_control (void)
{
struct  rccoord txtpos;
char buffer[100];

txtpos=_gettextposition();

use_control_window();
sprintf (buffer, "Control sp : %i min.", next_pfc_sp);
_settextposition(2,2);
_outtext(buffer);
sprintf (buffer, "Control nt : %i min.", next_pfc_nt);
_settextposition(3,2);
_outtext(buffer);
sprintf (buffer, "Control lq : %i min.", next_pfc_lq);
_settextposition(4,2);
_outtext(buffer);
sprintf (buffer, "Control rh : %i min.", next_pfc_rh);
_settextposition(5,2);
_outtext(buffer);

use_display_window();
_settextposition(txtpos.row,txtpos.col);

}
```

Module : Screen.c **Name :** InitialitzeSpPlot

Parameters : none

Output : none

Description : This function reads the configuration file for the graphical data representation of the spiruline compartment, and sets the internal variable properly.

Source code :

```
void InitializeSpPlot(void)
{
FILE *fp;
int i;
int v;

fp = fopen ("spgraph.cfg","rt");
if (fp==NULL)           /* If error, set 1 active flags to zero. */
    {
    spdata.activ[0] = 0;
    spdata.activ[1] = 0;
    spdata.activ[2] = 0;
    spdata.activ[3] = 0;
    spdata.activ[4] = 0;
    spdata.activ[5] = 0;
    spdata.activ[6] = 0;
    spdata.activ[7] = 0;
    spdata.activ[8] = 0;
    return;
    }
for (i=0; i < NUMVARSTOPLOT && ! feof (fp); i++)
    {
    v=fscanf (fp, "%i %i %f %f %15s", &spdata.activ[i], &spdata.colors[i],
&spdata.min[i], &spdata.max[i], spdata.name[i]);
    if (v != 5)
        {
        char buf[100];
        sprintf(buf, "Error in configuration file, line %i", i);
        display_error (buf);
        spdata.activ[0] = 0;
        spdata.activ[1] = 0;
        spdata.activ[2] = 0;
        spdata.activ[3] = 0;
        spdata.activ[4] = 0;
        spdata.activ[5] = 0;
        spdata.activ[6] = 0;
        spdata.activ[7] = 0;
        spdata.activ[8] = 0;
        i=NUMVARSTOPLOT;
        }
    }
```

```
fclose(fp);
}
```

Module : Screen.c **Name :** DrawLegend

Parameters : none

Output : none

Description : This function, accessing to the static pointer "actdata" - that points to the active data structure to plot - prints the legend in the bottom of the graphic.

Source code :

```
void DrawLegend (void)
{
int used=0,
    base=17,
    xbase = 0, i;


for (i=0; i < NUMVARSTOPLOT; i++)
    {
    if (actdata->activ[i])          /* Print in legend only active variables. */
        {
        use_display_window();
        _settextposition (base+used, 4+xbase);
        _settextcolor(actdata->colors[i]);
        _outtext(actdata->name[i]);
        _settextcolor(15);
        use_message_window();
        used++;
        if (used==3) { used = 0; base = 17; xbase +=17; }
        }
    }

use_message_window();

}
```

Module : Screen.c **Name :** DrawScale

Parameters : int number

Output : none

Description : Given the number of the wished variable to view, it prints the scale i the y axis. The colour used is the colour indicated in the structure pointed by actdata.

Source code :

```
void DrawScale(int i)
{
char buffer[100];

use_display_window();
_settextcolor(actdata->colors[i]);
_settextposition(3,0);
_outtext("      ");
_settextposition(3,0);
sprintf (buffer, "%4.2f", actdata->max[i]);
_outtext(buffer);
_settextposition(6,0);
_outtext("      ");
_settextposition(6,0);
sprintf (buffer, "%4.2f", 3*(actdata->max[i]/4));
_outtext(buffer);
_settextposition(9,0);
_outtext("      ");
_settextposition(9,0);
sprintf (buffer, "%4.2f", actdata->max[i]/2);
_outtext(buffer);
_settextposition(12,0);
_outtext("      ");
_settextposition(12,0);
sprintf (buffer, "%4.2f", actdata->max[i]/4);
_outtext(buffer);
_settextcolor(15);

_setcolor(15);
_moveto(Xo-3,Yo-Heigth);
_lineto(Xo+2,Yo-Heigth);

use_message_window();
}
```

Module : Screen.c **Name :** AdvanceScale

Parameters : none

Output : none

Description : This function is called when the plus key is pressed and graphical reperesentation is active, and its task is to select the next variable that is selectet to be plot.

Source code :

```
void AdvanceScale (void)
{
int i;

i = lastscale;

do
    {
    i = (i+1) % NUMVARSTOPLOT;
    }
    while (!actdata->activ[i] && i != lastscale );
lastscale = i;
DrawScale(lastscale);
}
```

Module : Screen.c **Name :** GoBackScale

Parameters : none

Output : none

Description : This function is called when the less key is pressed and graphical reperesentation is active, and its task is to select the previous variable that is selectet to be plot.

Source code :

```
void GoBackScale (void)
{
int i;

i = lastscale;

do
    {
    i = --i;
    if (i<0) i = NUMVARSTOPLOT-1;
    i = i % NUMVARSTOPLOT;
    }
    while (!actdata->activ[i] && i != lastscale );
lastscale = i;
DrawScale(lastscale);
}
```

Module : Screen.c **Name :** DrawAxes

Parameters : int numpoints, long begin, long end

Output : none

Description : This function is the responsible to draw the axes, print the label today, and the label of the first day wished for representation, indicated by the begin parameter. numpoints indicates the number of samples that are going to be plot.

Source code :

```
void DrawAxesSP (int np, long beg, long end)
{
char cad[100];

_setcolor (15);
_moveto (Xo, Yo);
_lineto (Xo+(Length*1.1), Yo);
_moveto (Xo, Yo);
_lineto (Xo, Yo - (Heigth *1.1));
_moveto (Xo-1, Yo+1);
_lineto (Xo+(Length*1.1), Yo+1);
_moveto (Xo-1, Yo);
_lineto (Xo-1, Yo - (Heigth *1.1));
_moveto(Xo-3,Yo - Heigth + (Heigth/2));
_lineto (Xo+2, Yo - Heigth + (Heigth/2) );
_moveto(Xo-3,Yo - Heigth + 3*(Heigth/4));
_lineto (Xo+2, Yo - Heigth + 3*(Heigth/4) );
_moveto(Xo-3,Yo - Heigth + (Heigth/4));
_lineto (Xo+2, Yo - Heigth + (Heigth/4) );
use_display_window();
_settextposition (16,4);
_settextcolor(11);
_outtext(olddate);
_settextposition (16,47);
_outtext ("today");
use_message_window();
}
```

Module : Screen.c **Name :** ReadDraw

Parameters : char *name, int np

Output : int, <0 if error ocurs.

Description : This function, given a file name (name) and a number of samples to plot (np) in total, reads the given file, and draws the graphic.

Source code :

```
int ReadDraw (char *name, int np)
{
FILE *fp;
char c[2], input[300];
char hora[6], cr;
int line=0;
int first=6;     /* N° of variables to graph */
double delta;     /* pixel per point */
int i;

double loc;


delta = Length / (double) np;

c[1]='\0';
strcpy (input, "");

_setcolor(15);
_moveto (x, Yo-2);
_lineto (x, Yo+3);

fp = fopen (name, "rt");
if (fp==NULL) {
                char buffer[100];
                sprintf(buffer, "Error opening file : %s", name);
                display_status(buffer);
                return -1;
              }
    else {
          read (&cr,1,1,fp);
          while (cr != '\n') fread (&cr, 1,1,fp);
          while (!feof(fp))
            {
            fread (&c[0], 1, 1, fp);
            if (ferror (fp)) {
                char buffer[100];
                sprintf(buffer, "Error opening file : %s", name);
                display_status(buffer);
                return -1;
              }
            if (c[0] == '\n')   /* let's read all a line ... */
```

```
                {
                line++;


                sscanf (input, "%5c %f %f %f %f %f %f %f %f",hora, &actdata-
>v[0], &actdata->v[1], &actdata->v[2], &actdata->v[3], &actdata->v[4], &actdata-
>v[5], &actdata->v[6], &actdata->v[7], &actdata->v[8]);
                for (i=0; i < 9; i++)
                  {
                    if (actdata->activ[i] ) {
                    if (!first) {
                                if ( Yo-Heigth < actdata->y[i] )
                                        _moveto (x,actdata->y[i]);
                                        else
                                        _moveto ((int)x+delta,Yo-Heigth);
                                }
                        else
                           {
                           if (Yo - (((actdata->v[i]-actdata->min[i])/actdata-
>max[i]) * Heigth)  > Yo - Heigth)
                                   {_moveto(x, Yo - (((actdata->v[i]-actdata-
>min[i])/actdata->max[i]) * Heigth) );}
                                  else
                                   {_moveto(x, Yo-Heigth); }
                           first--;
                           }
                    _setcolor (actdata->colors[i]);
                    actdata->y[i] = actdata->v[i];
                    actdata->y[i] = ((actdata->y[i]-actdata->min[i]) / actdata-
>max[i]) * Heigth;
                    actdata->y[i] = Yo - actdata->y[i];
                    if ( Yo-Heigth < actdata->y[i] )
                        _lineto ((int)x+delta,actdata->y[i]);
                        else
                        _lineto ((int)x+delta,Yo-Heigth);
                    }
                  }
                x += delta;
                strcpy (input, "");     /* Reinitialize input string */
                }
                else
                {
                strcat (input, c);
                }

            }

          fclose(fp);
          return line-2;    /* Return the n§ of points of the file (last, first)
*/
        }
}
```

Module : Screen.c **Name :** Draw

Parameters : int np, long begin, long end

Output : int, <0 if error, 0 in other case.

Description : Given two dates (begin and end) in japanese format (year-month-day), this function calculates the dates between them, calculate the name of the files to plot, and make them plot. The number of points to plot is used to be passed to ReadDrawSP (...)

Source code :

```
int DrawSP (int np, long beg, long end)
{
long k;
int r;
char cad[20];

x = Xo;


for (k=beg; k <= end; k++)
    {
    sprintf (cad, "sp%li.out", k);
    r = ReadDrawSP(cad,np);
    if (r==-1)
            {
            return -1;
            }
    }
return 0;

}
```

Module : Screen.c **Name :** CalculateNextDate

Parameters : long a

Output : long

Description : This function, given a date in japaneese format (year,month,day), calculate the next date and returns it.

Source code :

```
long CalculateNextDate(long a)
{
long  month ,year, day;

day = a %10;
a -= (a % 10);
a /= 10;
day += (a % 10)*10;

a -= (a % 10);
a /= 10;

month = a % 10;
a -= (a%10);
a /= 10;

month += (a % 10)*10;    /* Calculate the month */
a -= (a % 10);
a /= 10;

year = (a%10);
a -= (a %10);
a /= 10;
year += (a % 10)*10;

if (day==28 && month==2) return year*10000+(month+1)*100+1;
if ((day == 31) && ( month == 1 || month == 3 ||
                     month == 5 || month == 7 ||
                     month == 8 || month == 10 ) )
                        return year*10000 + (month+1)*100 + 1;
if ((day == 30) && ( month == 4 || month == 6 ||
                    month == 6 || month == 9 ||
                    month == 11 )) return year*10000 + (month+1)*100 + 1;

if (day==31 && month == 12) return (year+1)*10000+101;

return year*10000+month*100+day+1;
}
```

Module : Screen.c **Name :** testfileexist

Parameters : char *name

Output : int

Description : This function, given a file name, tests if it exist, if it does not returns -1, else it returns the number of lines dedicated to data that it contains.

Source code :

```
int testfileexist (char *name)
{
FILE *fp;
char c[2],
     hora[6],
     input[300];

int i;
int line=0;

c[1] = '\0';

strcpy (input, "");
fp = fopen (name, "rt");
if (fp==NULL) return -1;
    else {
          while (!feof(fp))
            {
            fread (&c[0], 1, 1, fp);
            if (c[0] == '\n')
                {
                line++;
                if (line >1)
                    {
                    /* Check for maximum */
                    sscanf (input, "%5c %f %f %f %f %f %f %f %f",hora,
                        &actdata->v[0], &actdata->v[1], &actdata->v[2], &actdata-
>v[3],
                        &actdata->v[4], &actdata->v[5], &actdata->v[6], &actdata-
>v[7],
                        &actdata->v[8]);
                    for (i=0; i < NUMVARSTOPLOT; i++)
                        {
                        if (actdata->v[i] > actdata->max[i]) actdata->max[i] =
actdata->v[i];
                        }
                    strcpy (input,"");
                    }
                }
                else {
                strcat(input, c);
```

```
                }
            }

            fclose(fp);
            return line-2;     /* Return the n§ of points of the file (last, first)
*/
        }
}
```

Module : Screen.c **Name :** calculate_between

Parameters : long begin, long end

Output : int

Description : Given two dates, it calculates the names of the files that are between them and test their existance. If all of them exist, it return the number of samples that will have to be plot, else it returns -1.

Source code :

```
int calcule_between (long beg, long end)
{
long k;
long lines=0, lf;
char cad[100], cad2[100];

_outtext ("\n\n");

for (k=beg; k <= end; k = CalculateNextDate (k))
    {
    sprintf (cad, "sp%li.out", k);
                /* Test if file exist */
    lf = (long) testfileexist(cad);
    if (lf==-1) {
                return -1;
                }
        else
        lines = lines + lf;
    }

return lines;
}
```

Module : Screen.c **Name :** use_status_window

Parameters : none

Output : none

Description : It sets as active text window the status region.

Source code :

```
use_status_window()
      {
     _settextwindow(24,2,24,55);
     tab(3,18);
       _settextcolor(WHITE);
       _setbkcolor((long)BLUE);
       _clearscreen(_GWINDOW);
       }
```

B.5 .- Module : Melfct

Module : melfct.c **Name :** wait_time

Parameters : int seconds

Output : none

Description : This function waits for "seconds" seconds, it is based on the routine timebase_main

Source code :

```
#ifndef ADERSA
wait_time(int i)
#else
wait_time( i)
int i;
#endif
      {
      char    buffer[100];
      unsigned long start_time;

      while(timebase_main()==0)
            {
            }
      start_time=timebase_main();
      while(timebase_main()<start_time+i)
            {
            }
      }
```

Module : melfct.c **Name :** timebase_main

Parameters : none

Output : time_t

Description : This is the routine that schedules the GPS, every minute, synchronised with the zero second of the real timer, this routine returns zero. In any other moment, it returns the actual time.

Source code :

```
timebase_main(void)
      {
      time_t  ltime;
      char    buffer[100];
      struct  tm  *dt;
      double  minute;

      time(&ltime);
    if(main_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);
       main_last_display=ltime;
           }

      time(&ltime);
    if (main_lastsamp==0)
           {
           display_status("timebase synchronisation ...");
       minute= TSAMP_MAIN / 60;
           while((ceil(dt->tm_min/minute)!=dt->tm_min/minute)||(dt->tm_sec!=0))
                 {
                 time(&ltime);
                 dt=localtime(&ltime);
                 dt->tm_mon++;
           if(main_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);
                main_last_display=ltime;
                       }
                 }
        main_lastsamp=ltime;
           display_status(" ");
           return(0);
```

```
        }
if ((main_lastsamp+main_tsamp)<=ltime)
        {
    main_lastsamp=main_lastsamp+main_tsamp;
        return(0);
        }
  return(ltime);
  }
```

B.6 .- Module : Results

Module : results.c **Name :** result_reactor3

Parameters : none

Output : none

Description : This function has no functionality yet, but will be renamed and completed to allow printing the rhodobacter compartment.

Source code :

```
result_reactor3()
{
#ifndef ADERSA
        void display_result(char *,short,short);
#else
        void display_result();
#endif
        char buffer[150];

    set_title ("Reactor III");

    use_display_window();
    _clearscreen(_GWINDOW);
    use_message_window();


    sprintf(buffer,"Data not ready.");
    display_result(buffer,20,5);

    sprintf(buffer,"F1 - Help");
    center_display_result(buffer,20);
}
```

Module : results.c **Name :** log_file

Parameters : char *input

Output : none

Description : This function sends the input string to a log file called "log.txt".

Source code :

```
void log_file (char *buffer)
{
FILE *fp;
int lng;
char bf[200];
time_t ltime;
struct  tm  *dt;

if (diskfull) return;

fp = fopen ("log.txt", "at");
if (fp==NULL) { use_message_window();
                _outtext("Couldn't open log file.\n");
                use_display_window();
              }
              else {
                time(&ltime);
                dt=localtime(&ltime);
                dt->tm_mon++;
                sprintf(bf," %02d/%02d/%02d   %02d:%02d:%02d Msg : ",dt->tm_mday,
                dt->tm_mon,dt->tm_year,dt->tm_hour,dt->tm_min,dt->tm_sec);

                strcat (bf, buffer);
                strcat(bf,"\n");
                lng = strlen(bf);
                fwrite (bf, 1, lng, fp);
                fclose(fp);
              }
}
```

Module : results.c **Name :** error_log_file

Parameters : char *input

Output : none

Description : Its task is to send a message to the log file, but labeling it as an error, also date and time appear.

Source code :

```
void error_log_file (char *buffer)
{
FILE *fp;
int lng;
char bf[200];
time_t ltime;
struct  tm  *dt;

if (diskfull) return;

fp = fopen ("log.txt", "at");
if (fp==NULL) { use_message_window();
                _outtext("Couldn't open log file.\n");
                use_display_window();
              }
              else {
                  time(&ltime);
                  dt=localtime(&ltime);
                  dt->tm_mon++;
                  sprintf(bf," %02d/%02d/%02d   %02d:%02d:%02d Error : ",dt-
>tm_mday,
                  dt->tm_mon,dt->tm_year,dt->tm_hour,dt->tm_min,dt->tm_sec);

                  strcat (bf, buffer);
                  strcat(bf,"\n");
                  lng = strlen(bf);
                  fwrite (bf, 1, lng, fp);
                  fclose(fp);
              }
}
```

Module : results.c **Name :** result

Parameters : ScreenView *display, int new

Output : none

Description : This function, depending on the value of the pointer "display", selects a screen to print. When the selection is made, this routine calls another that is going to have the task of printing the screen.

Source code :

```
int result(ScreenViews *display_reactor, int new)
{
#ifndef __Monitoring
        next_control ();
#endif

switch(*display_reactor)     /* Choose the info to show */
    {
    case help_1 : help();
                  break;
    case filemanagement : file_menu();
                  break;
    case principal : result_principal();
                     break;
    case spirulina :  result_spirulina();
                      break;
    case gspirulina : SelectSpirulineGraph ();
                      if (!drawgraph(new,0)){
                                          *display_reactor=principal;
                                          return 0;
                                          }
                      break;
    case igspirulina : if (new) SelectSpirulineGraph ();
                      if (!drawgraph(new,1)) {
                                           *display_reactor=principal;
                                           return 0;
                                           }
                      break;
    case reactor_2 : result_nitrogen();
                     break;
    case reactor_3 : result_reactor3();
                     break;
    case reactor_4 : result_reactor4();
                     break;
    case status : result_status();
                  break;
    default : break;
    }
return 1;
}
```

Module : results.c **Name :** result_spirulina

Parameters : none

Output : none

Description : This routine prints the values of the variables for the spirulina compartment.

Source code :

```
result_spirulina()
        {
#ifndef ADERSA
        void display_result(char *,short,short);
#else
        void display_result();
#endif
        char buffer[150];

        set_title ("Spirulina compartment");

        use_display_window();
        _clearscreen(_GWINDOW);
        use_message_window();


        display_result ("Value", 27, 2);
        display_result ("SetPoint", 38, 2);

        _moveto (30, 100);
        _lineto (375,100);
        _moveto (190, 80);
        _lineto (190, 190);
        _moveto (280, 80);
        _lineto (280, 190);

        display_result("Biomass (mg/l)",7,4);
        display_result("Nitrate (mg/l)",7,5);
        sprintf(buffer,"%.1f",cxa.value);
        display_result(buffer,27,4);
        sprintf(buffer,"%.1f",nitrate.value);
        display_result(buffer,27,5);
        sprintf(buffer,"%.1f",cxa.sp);
        display_result(buffer,39,4);
        sprintf(buffer,"%.1f",nitrate.sp);
        display_result(buffer,39,5);

        display_result("Eb      (W/m2) ",7,6);
        sprintf(buffer,"%.1f", Eb.value);
        display_result(buffer,27,6);
        sprintf(buffer,"%.1f", Eb.sp);
        display_result(buffer,39,6);
```

```
display_result("Fr      (W/m2) ",7,7);
sprintf(buffer,"%.1f", Fr.value);
display_result(buffer,27,7);
sprintf(buffer,"%.1f", Fr.sp);
display_result(buffer,39,7);


display_result ("Setpt", 21,12);
display_result ("Model", 29,12);
display_result ("Realised",36,12);
display_result ("Mesured", 45, 12);

_moveto (280, 230);
_lineto (280, 320);
_moveto (221, 230);
_lineto (221, 320);
_moveto (160, 230);
_lineto (160, 320);
_moveto (356, 230);
_lineto (356, 320);
_moveto (10, 260);
_lineto (430,260);

display_result("Productivity (mg/h)",1,14);

sprintf(buffer,"%.2f",cons_prod_nom.value); /* SetPoint */
display_result(buffer,21,14);

sprintf(buffer,"%.2f",prod_mod.sp);         /* Model */
display_result(buffer,29,14);

sprintf(buffer,"%.2f",cons_prod_real.sp);   /* Realised */
display_result(buffer,37,14);


sprintf(buffer,"%.2f",production.sp);   /* Mesured */
display_result(buffer,46,14);


display_result("Flow         (l/h)", 1,15);
sprintf(buffer,"%.3f",qe_real.sp);
display_result(buffer,37,15);           /* Realised */

sprintf(buffer,"%.3f",qe_real.value);
display_result(buffer,21,15);           /* Setpoint */

}
```

Module : results.c **Name :** DrawCompartment

Parameters : int x0, int y0, int x1, int y1

Output : none

Description : This routine draws a box that is said to be a compartment for the main screen.

Source code :

```
void DrawCompartment(int x, int y)
{
_moveto(x,y);
_lineto(x+60,y);
_lineto(x+60,y+40);
_lineto(x,y+40);
_lineto(x,y);
}
```

Module : results.c **Name :** DownArrow

Parameters : int x, int y

Output : none

Description : This routine draws an arrow pointing down.

Source code :

```
void DownArrow (int x, int y)
{
_moveto(x,y);
_lineto(x-3,y-3);
_moveto(x,y);
_lineto(x+3,y-3);
}
```

Module : results.c **Name :** UpArrow

Parameters : int x, int y

Output : none

Description : This routine draws an arrow pointing up.

Source code :

```
void UpArrow (int x, int y)
{
_moveto(x,y);
_lineto(x-3,y+3);
_moveto(x,y);
_lineto(x+3,y+3);
}
```

Module : results.c **Name :** LeftArrow

Parameters : int x, int y

Output : none

Description : This routine draws an arrow pointing left.

Source code :

```
void UpArrow (int x, int y)
{
_moveto(x,y);
_lineto(x-3,y+3);
_moveto(x,y);
_lineto(x+3,y+3);
}
```

Module : results.c **Name :** RightArrow

Parameters : int x, int y

Output : none

Description : This routine draws an arrow pointing right.

Source code :

```
void RightArrow (int x, int y)
{
_moveto(x,y);
_lineto(x-3,y-3);
_moveto(x,y);
_lineto(x-3,y+3);
}
```

Module : results.c **Name :** tech_log_file

Parameters : char *input

Output : none

Description : This function sends the input string to a technical log file, used for testing and named "techlog.txt". Also, in the technical log apears the time and date.

Source code :

```
void tech_log_file (char *buffer)
{
FILE *fp;
int lng;
char bf[200];
time_t ltime;
struct  tm  *dt;

if (diskfull) return;

fp = fopen ("techlog.txt", "at");
if (fp==NULL) { use_message_window();
                _outtext("Couldn't open log file.\n");
                use_display_window();
              }
              else {
                time(&ltime);
                dt=localtime(&ltime);
                dt->tm_mon++;
                sprintf(bf," %02d/%02d/%02d   %02d:%02d:%02d Msg : ",dt->tm_mday,
                dt->tm_mon,dt->tm_year,dt->tm_hour,dt->tm_min,dt->tm_sec);

                strcat (bf, buffer);
                strcat(bf,"\n");
                lng = strlen(bf);
                fwrite (bf, 1, lng, fp);
                fclose(fp);
              }
}
```

Module : results.c **Name :** result_nitrogen

Parameters : none

Output : none

Description : This function prints the results of the nitrogen compartment, actualy its information is just for testing.

Source code :

```
result_nitrogen()
{
#ifndef ADERSA
        void display_result(char *,short,short);
#else
        void display_result();
#endif
        char buffer[150];

    set_title ("Reactor II");

    use_display_window();
    _clearscreen(_GWINDOW);
    use_message_window();


        sprintf(buffer,"LOOP0303 %f", AI0301.value);
        display_result(buffer,2,2);
        sprintf(buffer,"LOOP0307 %f", AI0302.value);
        display_result(buffer,1,3);
        sprintf(buffer,"AI--0303 %f", AI0303.value);
        display_result(buffer,1,4);
        sprintf(buffer,"AI-0304 %f", AI0304.value);
        display_result(buffer,1,5);
        sprintf(buffer,"AI-0305 %f", AI0305.value);
        display_result(buffer,1,6);
        sprintf(buffer,"LOOP0304 %f", AI0306.value);
        display_result(buffer,1,7);
        sprintf(buffer,"LOOP0305 %f", AI0307.value);
        display_result(buffer,1,8);
        sprintf(buffer,"LOOP0306 %f", AI0308.value);
        display_result(buffer,1,9);


    sprintf(buffer,"F1 - Help");
    center_display_result(buffer,20);
}
```

Module : results.c **Name :** directory

Parameters : none

Output : none

Description : Its task is to get the name and size of the files ending in .txt and .out. After that those files are printed in the screen.

Source code :

```
void directory(void)
{
#ifndef ADERSA
        void display_result(char *,short,short);
#else
        void display_result();
#endif
        char buffer[150];
        struct find_t fi;
        int x=3, y=3;
        struct diskfree_t df;
        unsigned long lliure;

    set_title ("Directory");

    use_display_window();
    _clearscreen(_GWINDOW);
    use_message_window();


    if (_dos_findfirst("*.out",_A_NORMAL,&fi) == 0)
        {
        sprintf (buffer, "%- 12s % 6i bytes", fi.name, fi.size);
        display_result(buffer, x, y);
        while (_dos_findnext(&fi)==0)
            {
            if ((++y) == 20) {
                                    display_status ("Press a key ...");
                                    while (!kbhit());
                                    getch();
                                    use_display_window();
                                    _clearscreen(_GWINDOW);
                                    use_message_window();
                                    sprintf(buffer,"F1 - Help");
                                    center_display_result(buffer,20);
                                    x = 3; y = 3;
                                }
            sprintf (buffer, "%- 12s % 6i bytes \n", fi.name, fi.size);
            display_result(buffer, x, y);
            }
        }
```

```
        if ((++y) == 20) {
                                display_status ("Press a key ...");
                                while (!kbhit());
                                getch();
                                use_display_window();
                                _clearscreen(_GWINDOW);
                                use_message_window();
                                sprintf(buffer,"F1 - Help");
                                center_display_result(buffer,20);
                                x = 3; y = 3;
                        }


    if (_dos_findfirst("*.txt",_A_NORMAL,&fi) == 0)
        {
        sprintf (buffer, "%- 12s % 6i bytes", fi.name, fi.size);
        display_result(buffer, x, y);
        while (_dos_findnext(&fi)==0)
            {
            if ((++y) == 20) {
                                    display_status ("Press a key ...");
                                    while (!kbhit());
                                    getch();
                                    use_display_window();
                                    _clearscreen(_GWINDOW);
                                    use_message_window();
                                    sprintf(buffer,"F1 - Help");
                                    center_display_result(buffer,20);
                                    x = 3; y = 3;
                                }
            sprintf (buffer, "%- 12s % 6i bytes \n", fi.name, fi.size);
            display_result(buffer, x, y);
            }
        }

    _dos_getdiskfree(0, &df);   /* Default drive. */
    lliure = (unsigned long) df.avail_clusters *
                (unsigned long)  df.sectors_per_cluster *
                (unsigned long) df.bytes_per_sector;


    sprintf (buffer,"*.out, *.txt %li bytes free", lliure);
    display_status (buffer);
    sprintf(buffer,"F1 - Help");
    center_display_result(buffer,20);
}
```

Module : results.c **Name :** result_reactor4

Parameters : none

Output : none

Description : This function has no functionality yet, but will be renamed and completed to allow printing the liquefying compartment.

Source code :

```
result_reactor4()
{
#ifndef ADERSA
        void display_result(char *,short,short);
#else
        void display_result();
#endif
        char buffer[150];

    set_title ("Reactor IV");

    use_display_window();
    _clearscreen(_GWINDOW);
    use_message_window();


    sprintf(buffer,"Data not ready.");
    display_result(buffer,20,5);

    sprintf(buffer,"F1 - Help");
    center_display_result(buffer,20);
}
```

Module : results.c **Name :** init_output_file

Parameters : FILE *fp, char *name

Output : int != if error, ==0 else.

Description : This routine is called when the GPS is started and its task is to check if is there any problem for having an output file, if it does not exist this function creates one and insert in it a header.

Source code :

```
int init_output_file(FILE *fp, char *name)
{
if (!diskfull)
use_message_window();
_clearscreen(_GWINDOW);
_outtext("Initializing output files ...");
if (!diskfull)
    {
    fp=fopen(name,"rt");
    if ( fp == NULL)
            {
            fp=fopen(name,"wt");
            if ( fp == NULL)
                    { display_error ("Impossible to open output file !!");
                      wait_time(1);
                     return 1;
                    }
                    else
                    {
                    char buffer[400];
                    sprintf (buffer, "cxa_moy, nit_moy, production.sp,
react.temp, cons_prod_real.sp,  qe_real.sp, qe_real.value, Fr.sp, Eb.sp\n");
                    fwrite(buffer, 1, strlen(buffer), fp);
                    }
            }
        _outtext(" done.\n");
    }
    else _outtext (" canceled\n");
   _outtext(" done.\n");
   wait_time(1);
   _clearscreen (_GWINDOW);
   fclose (fp);
   return 0;
}
```

Module : results.c **Name :** init_log_file

Parameters : none

Output : none

Description : This function is called when the GPS is started and send to the log file the string REINITIALIZING. This way we can realise how many times the GPS has been reinitialized. This was made due to some electrical problems.

Source code :

```
int init_log_file(void)
{
FILE *fp;
char bf[200];
time_t ltime;
struct  tm  *dt;


use_message_window();
_clearscreen(_GWINDOW);
_outtext("Initializing log file ...");

if (!diskfull)
    {
    fp=fopen("log.txt","at");
    if ( fp == NULL)
            {
            display_error ("Impossible to open output file !!");
            wait_time(1);
            return 1;
            }
       time(&ltime);
       dt=localtime(&ltime);
       dt->tm_mon++;
       sprintf(bf," %02d/%02d/%02d   %02d:%02d:%02d Msg : ",dt->tm_mday,
              dt->tm_mon,dt->tm_year,dt->tm_hour,dt->tm_min,dt->tm_sec);
       strcat (bf, "REINITIALIZING\n");
       fwrite(bf, 1, strlen(bf), fp);
       wait_time(1);
       fclose (fp);
    fp=fopen("techlog.txt","at");
    if ( fp == NULL)
            {
            display_error ("Impossible to open technical output file !!");
            wait_time(1);
            return 1;
            }
       fwrite(bf,1,strlen(bf), fp);
```

```
        _outtext(" done.\n");
     }
     else _outtext(" canceled.\n");
    wait_time(1);
    fclose (fp);
    _clearscreen (_GWINDOW);
    return 0;
}
```

Module : results.c **Name :** result_status

Parameters : none

Output : none

Description : This function prints the screen that shows the user a system status. The information showed is coeficient alpha of filters, sampling period and disk status.

Source code :

```
result_status()
{
#ifndef ADERSA
        void display_result(char *,short,short);
#else
        void display_result();
#endif
        char buffer[150];

    set_title ("System status");


    use_display_window();
    _clearscreen(_GWINDOW);
    use_message_window();

    if (ALPHAFILTERcxa!=1) sprintf(buffer,"Cxa filter : %.2f", ALPHAFILTERcxa);
        else sprintf(buffer,"Cxa filter : OFF");
    display_result(buffer,2,2);
    if (ALPHAFILTERnitrate!=1) sprintf(buffer,"Nitrate filter : %.2f",
ALPHAFILTERnitrate);
        else sprintf(buffer,"Nitrate filter : OFF");
    display_result(buffer,2,3);
    if (ALPHAFILTEREb!=1) sprintf(buffer,"Eb filter : %.2f", ALPHAFILTEREb);
        else sprintf(buffer,"Eb filter : OFF");
    display_result(buffer,2,4);
    if (ALPHAFILTERpH != 1) sprintf(buffer,"Ph filter : %.2f", ALPHAFILTERpH);
        else sprintf(buffer,"Ph filter : OFF");
    display_result(buffer,2,5);

    sprintf(buffer,"Sampling sp : %i m.", SAMP_SPIRU);
    display_result(buffer,2,7);

    sprintf(buffer,"Sampling lq : %i m.", SAMP_LIQUE);
    display_result(buffer,2,8);

    sprintf(buffer,"Sampling rh : %i m.", SAMP_RHODO);
    display_result(buffer,2,9);

    sprintf(buffer,"Sampling nt : %i m.", SAMP_NITRI);
    display_result(buffer,2,10);
```

```
    sprintf (buffer, "Alarms : ");
    if (activated_alarms) strcat (buffer, "ON");
       else strcat (buffer, "OFF");
    display_result(buffer,35,2);

    if (diskwarning) {
            strcpy (buffer,"WARNING !! DISK TOO FULL");
            display_result(buffer,35,3);
            }
            else
            {
            strcpy (buffer,"Disk status : Ok.");
            display_result(buffer,35,3);
            }
    if (diskfull)
        {
        strcpy (buffer,"WARNING !! DISK FULL");
        display_result(buffer,35,4);
        strcpy (buffer,"WRITING DISABLED");
        display_result(buffer,39,5);
        }

    sprintf(buffer,"F1 - Help");
    center_display_result(buffer,20);
}
```

Module : results.c **Name :** move_disk

Parameters : none

Output : none

Description : This routine task is to ask the user about what file to move, calculate the new name, and test if the name introduced is in the group of permited files or not. After this and a safety test, the moving routine is called.

Source code :

```
void move_disk (void)
{
char buffer[150];
char name[40], destination[40];
unsigned long lliure;

set_title ("Floppy backups");

use_display_window();
_clearscreen(_GWINDOW);
use_message_window();

sprintf(buffer,"Please, insert a floppy disk in drive A: and ");
center_display_result(buffer,3);
sprintf(buffer,"type the name of the file you want to backup.");
center_display_result(buffer,4);

sprintf(buffer,"The syntax to use is : xxyymmdd where ...");
center_display_result(buffer,6);
sprintf(buffer,"xx is the reactor name sp, nt, lq or rh.");
center_display_result(buffer,7);
sprintf(buffer,"yy is the year.");
center_display_result(buffer,8);
sprintf(buffer,"yy is the month.");
center_display_result(buffer,9);
sprintf(buffer,"yy is the day.");
center_display_result(buffer,10);

sprintf(buffer,"F1 - Help");
center_display_result(buffer,20);

sprintf(buffer,"File name : ");
display_result(buffer,15,13);

strcpy (buffer, "");
get_input_string (name, 27 ,13);
if (strlen (name) > 8 || strlen (name) == 0
```

```
    || !(toupper (name[0]) == 'S' && toupper (name[1]) == 'P') )
                    {
                    display_status ("This is not a valid file name");
                    printf("\a\a");
                    wait_time(2);
                    display_status (" ");
                    return;
                    }

sprintf (buffer, "%s.out",name);      /* Complete the file name */
sprintf (destination, "a:%s",buffer);
strcpy (name, buffer);

move_files (name, destination);


display_status (" ");
}
```

Module : results.c **Name :** copy_disk

Parameters : none

Output : none

Description : This routine task is to ask the user about what file to copy, calculate the new name, and test if the name introduced is in the group of permited files or not. After this and a safety test, the copying routine is called.

Source code :

```
void copy_disk (void)
{
char buffer[150];
char name[40], destination[40];
unsigned long lliure;

set_title ("Floppy backups");

use_display_window();
_clearscreen(_GWINDOW);
use_message_window();

sprintf(buffer,"Please, insert a floppy disk in drive A: and ");
center_display_result(buffer,3);
sprintf(buffer,"type the name of the file you want to backup.");
center_display_result(buffer,4);

sprintf(buffer,"The syntax to use is : xxyymmdd where ...");
center_display_result(buffer,6);
sprintf(buffer,"xx is the reactor name sp, nt, lq or rh.");
center_display_result(buffer,7);
sprintf(buffer,"yy is the year.");
center_display_result(buffer,8);
sprintf(buffer,"yy is the month.");
center_display_result(buffer,9);
sprintf(buffer,"yy is the day.");
center_display_result(buffer,10);

sprintf(buffer,"F1 - Help");
center_display_result(buffer,20);

sprintf(buffer,"File name : ");
display_result(buffer,15,13);

strcpy (buffer, "");
get_input_string (name, 27 ,13);
if (strlen (name) > 8 || strlen (name) == 0
    || !(toupper (name[0]) == 'S' && toupper (name[1]) == 'P') )
```

```
                    {
                    display_status ("This is not a valid file name");
                    printf("\a\a");
                    wait_time(2);
                    display_status (" ");
                    return;
                    }

sprintf (buffer, "%s.out",name);      /* Complete the file name */
sprintf (destination, "a:%s",buffer);
strcpy (name, buffer);


lliure = checkfloppy();
if (lliure == -1) {
                  display_status ("Drive A: is not ready.");
                  printf ("\a\a");
                  wait_time(2);
                  display_status (" ");
                  return;
                  }

if (copia(name, destination, lliure)!=0)
        {
        display_status ("An error occurred while copying ...");
        printf ("\a\a");
        }

display_status (" ");
}
```

Module : results.c **Name :** file_menu

Parameters : none

Output : none

Description : This function prints the file managment menu

Source code :

```
void file_menu(void)
{
#ifndef ADERSA
        void display_result(char *,short,short);
#else
        void display_result();
#endif
        char buffer[150];

    set_title ("File options");

    use_display_window();
    _clearscreen(_GWINDOW);
    use_message_window();

    sprintf(buffer,"F5 - Move to floppy output files.");
    display_result(buffer,20,5);
    sprintf(buffer,"F6 - Copy to floppy log file.");
    display_result(buffer,20,6);
    sprintf(buffer,"F7 - Remove log file.");
    display_result(buffer,20,7);
    sprintf(buffer,"F8 - Directory.");
    display_result(buffer,20,8);
    sprintf(buffer,"F9 - Move to floppy output files.");
    display_result(buffer,20,9);

    sprintf(buffer,"F1 - Help");
    center_display_result(buffer,20);
}
```

Module : results.c **Name :** result_principal

Parameters : none

Output : none

Description : This routine prints the global information of the plant, the information that does not belong to any compartment in particular.

Source code :

```
result_principal()
{
#ifndef ADERSA
        void display_result(char *,short,short);
#else
        void display_result();
#endif
        char buffer[150];

    set_title ("Main screen");

    use_display_window();
    _clearscreen(_GWINDOW);

_settextcolor(12);
_settextposition (5+TVA,37+THA);
_outtext ("Biomass");
_settextposition (15+TVA,42+THA);
_outtext ("Waste");

_settextcolor(2);
_settextposition (5+TVA,20+THA);
_outtext ("O2");
_settextposition (15+TVA,25+THA);
_outtext ("CO2");

_settextcolor(15);

_setcolor (2);
_moveto(200+HA,80+VA);
_lineto(60+HA,80+VA);
_lineto (60+HA, 164+VA);
DownArrow (60+HA,163+VA);

_setcolor (12);
_moveto(200+HA,100+VA);
_lineto(80+HA,100+VA);
_lineto (80+HA, 164+VA);
RightArrow (199+HA,100+VA);

_setcolor (12);
```

```
_moveto (110+HA,300+VA);
_lineto(70+HA,300+VA);
_lineto(70+HA,205+VA);
UpArrow(70+HA,205+VA);

_setcolor (2);
_moveto (130+HA,280+VA);
_lineto(130+HA,220+VA);
_lineto(220+HA,220+VA);
_lineto (220+HA,110+VA);
UpArrow(220+HA,111+VA);

_setcolor (12);
_moveto (150+HA,280+VA);
_lineto(150+HA,190+VA);
_lineto(340+HA,190+VA);
RightArrow(339+HA,190+VA);


_setcolor (2);
_moveto (290+HA, 300+VA);
_lineto (220+HA,300+VA);
_lineto (220+HA, 220+VA);
UpArrow(220+HA,221+VA);

_setcolor (12);
_moveto(320+HA, 320+VA);
_lineto(320+HA, 340+VA);
_lineto(140+HA,340+VA);
_lineto(140+HA,320+VA);
UpArrow(140+HA,321+VA);

_setcolor (12);
_moveto(370+HA,204+VA);
_lineto(370+HA, 240+VA);
_lineto(320+HA, 240+VA);
_lineto(320+HA,280+VA);
DownArrow(320+HA,279+VA);

_setcolor (2);
_moveto (260+HA,100+VA);
_lineto (360+HA,100+VA);
_lineto(360+HA, 164+VA);
LeftArrow (261+HA,100+VA);

_setcolor (2);
_moveto (240+HA,110+VA);
_lineto (240+HA,174+VA);
_lineto(340+HA, 174+VA);
RightArrow (340+HA,174+VA);

_setcolor (12);
_moveto (260+HA,80+VA);
_lineto (380+HA,80+VA);
_lineto(380+HA, 164+VA);
DownArrow (380+HA,163+VA);

_setcolor(15);
```

```
DrawCompartment (200+HA, 70+VA);
DrawCompartment (41+HA, 164+VA);
DrawCompartment (341+HA, 164+VA);
DrawCompartment (110+HA, 280+VA);
DrawCompartment (290+HA, 280+VA);

_settextcolor (15);
_settextposition (6+TVA,29+THA);
_outtext ("iv");
_settextposition (12+TVA,9+THA);
_outtext ("iii");
_settextposition (12+TVA,46+THA);
_outtext ("CREW");
_settextposition (19+TVA,18+THA);
_outtext ("ii");
_settextposition (19+TVA,41+THA);
_outtext ("i");

_settextcolor (12);
_settextposition (9+TVA,10+THA);
_outtext ("NO3-");
_settextposition (17+TVA,9+THA);
_outtext ("NH4-");
_settextposition (21+TVA,27+THA);
_outtext ("Acids");
_settextposition (13+TVA,33+THA);
_outtext ("Biomass");

_settextcolor(2);
_settextposition (10+TVA,45+THA);
_outtext ("CO2");
_settextposition (8+TVA,30+THA);
_outtext ("CO2");

_settextcolor (15);

    use_message_window();

    sprintf(buffer,"F1 - Help");
    center_display_result(buffer,20);

}
```

B.7 .- Module : Gpsfile.

Module : gpsfile.c **Name :** close_grp_gps

Parameters : none

Output : none

Description : This routine closes all the files in the ring of GPS_FILEs. Finaly it calls the function "display_no_active_group".

Source code :

```
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();
      }
```

Module : gpsfile.c **Name :** active_grp_gps

Parameters : none

Output : GPS_FILE*

Description : This function, sets the next GPS_FILE in the ring, and returns a pointer to it. If an error ocurs, the program is stopped.

Source code :

```
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);
      }
```

Module : gpsfile.c **Name :** open_file_gps

Parameters : none

Output : GPS_FILE*

Description : This function reads the file "GPS.FIL" and open the files that are indicated in the first. Finaly it returns a pointer to the ring of GPS_FILE structures.

Source code :

```
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++)
                  {
                  if (strlen (str)>6) {
                                    sprintf(buffer,"%s.GPS",str);
                                    if (i>1) break;
                                    }
                        else
                        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);
      }
```

B.8 .- Module : Help

Module : Help.c **Name :** help

Parameters : none

Output : none

Description : This function prints the help screen.

Source code :

```
void help (void)
{
    use_display_window();
    _clearscreen(_GWINDOW);
    use_message_window();


    sprintf(buffer,"*** MELISSA - Help ***");
    display_result(buffer,28,1);

    sprintf(buffer,"F1 ... Help.");
    display_result(buffer,20,4);
    sprintf(buffer,"ALT+F1 ... Repaint screen.");
    display_result(buffer,20,5);
    sprintf(buffer,"F2 ... System status.");
    display_result(buffer,20,6);
    sprintf(buffer,"F3 ... File management menu.");
    display_result(buffer,20,7);
    sprintf(buffer,"F5 ... Spirulina reactor.");
    display_result(buffer,20,8);
    sprintf(buffer,"ALT+F5 ... Spirulina reactor evolution.");
    display_result(buffer,20,9);
    sprintf(buffer,"CTR+F5 ... Spirulina reactor interval.");
    display_result(buffer,20,10);
    sprintf(buffer,"F6 ... Reactor 2.");
    display_result(buffer,20,11);
    sprintf(buffer,"F7 ... Reactor 3.");
    display_result(buffer,20,12);
    sprintf(buffer,"F8 ... Reactor 4.");
    display_result(buffer,20,13);
    sprintf(buffer,"Esc .. Main screen.");
    display_result(buffer,20,15);
}
```

B.9 .- Module : Ctrlspir

Module : ctrlspir.c **Name :** average_var

Parameters : VARS *diff_var, int diff_time

Output : double

Description : This routine returns the average during time diff_time (in minutes) of the variable pointed by diff_var.

Source code :

```
#ifndef ADERSA
double average_var(VARS *diff_var, int diff_time)
#else
double average_var(diff_var, diff_time)
VARS *diff_var;
int diff_time;
#endif
      {
      int j;
      int i_samp,nb_samp;
      double average;

      average=0;
    nb_samp=ceil(diff_time*60/(TSAMP_MAIN * SAMP_SPIRU));
      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);
      }
```

Module : ctrlspir.c **Name :** copy_react

Parameters : REACT *reacta, REACT *reactb

Output : none

Description : It copies the reacta structure to the reactb structure.

Source code :

```
copy_react(REACT *reacta,REACT *reactb)
#else
copy_react(reacta,reactb)
REACT *reacta;
REACT *reactb;
#endif

      {
      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;
      }
```

Module : ctrlspir.c **Name :** init_vars_spirulina

Parameters : none

Output : none

Description : This is the function that initializes the variables for the spirulina compartment.

Source code :

```
init_vars_spirulina()
      {
      REACT   init_react;
      double  delta;
      int     jj;

      first_time_spiruline = 1;          /* Set for the first filter entry */
      display_status("Initialisation of variables ...");

      /* Initialisation of filter parameters */


/*  TAG and COMMAND name initialisation     */

      sprintf(cxa.name,"LOOP0107");
      sprintf(nitrate.name,"LOOP0103");
      sprintf(cal_nitrate.name,"DI--0125");

      sprintf(Eb.name,"LOOP0105");
      sprintf(Fr.name,"LOC-0128");
      sprintf(temperature.name,"LOOP0106");
      sprintf(pH.name,"LOOP0104");
      sprintf(act_pompe.name,"LOC-0154");
      sprintf(cal_pump.name,"LOC-0137");

      sprintf(cons_prod_nom.name,"LOC-0150");
      sprintf(cons_prod_real.name,"LOC-0152");
      sprintf(qe_nom.name,"LOC-0151");
      sprintf(qe_real.name,"LOC-0153");
      sprintf(production.name,"LOC-0155");
      sprintf(prod_mod.name,"LOC-0156");

/*  Variables initialisation   */

      acq_vars_spirulina();

      init_react.Cxa=cxa.value;
      init_react.Cno3=nitrate.value;
      init_react.temp=temperature.value;
      init_react.Eb=Eb.value;
      model(&init_react,USE_EB);
      Fr.sp=init_react.Fr;
```

```
      write_var(&Fr);
      fill_struct_var(&cxa);

      cons_prod_real.sp=cons_prod_nom.value;
      write_var(&cons_prod_real);
      qe_real.sp=qe_nom.value;
      write_var(&qe_real);

/* initialisation timer PFC */
      next_pfc_sp=DT;

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

Module : ctrlspir.c **Name :** acq_vars_spirulina

Parameters : none

Output : none

Description : This function reads the variables of the compartment. After reading them the filter function is called when is necesary.

Source code :

```
acq_vars_spirulina()
      {
      char bf[200];
      double realval;

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

    read_var(&cxa);
    if (!first_time_spiruline)
          {
        filter(cxa.name);
          }

/*  nitrate analyser calibration  */

      read_var(&cal_nitrate);
      if(!cal_nitrate.value)
          {
          read_var(&nitrate);
        if (!first_time_spiruline)
              {
          filter(nitrate.name);
              }
          }

      read_var(&Eb);
    if (!first_time_spiruline)
          {
        filter(Eb.name);
          }
      read_var(&Fr);
      read_var(&temperature);
    read_var(&pH);
    if (!first_time_spiruline)
          {
        filter(pH.name);
          }
      read_var(&act_pompe);
    read_var(&cal_pump);

      read_var(&cons_prod_nom);
```

```
read_var(&cons_prod_real);
read_var(&qe_nom);
read_var(&qe_real);
read_var(&production);
read_var(&prod_mod);


display_status(" ");

  if (first_time_spiruline) first_time_spiruline = 0;
  }
```

Module : ctrlspir.c **Name :** send_vars_spirulina

Parameters : none

Output : none

Description : This routine sends the variables to be send for the spirulina compartment.

Source code :

```
send_vars_spirulina()
      {
#ifndef __Monitoring
    display_status("Updating variables ...");


      write_var(&Eb);
      write_var(&Fr);
      write_var(&act_pompe);
      write_var(&cons_prod_real);
      write_var(&qe_real);
      write_var(&production);
      write_var(&prod_mod);

#endif
      display_status(" ");
      }
```

Module : ctrlspir.c **Name :** model

Parameters : REACT *react, int mode

Output : none

Description : This is the mathematical model for the spirulina compartment.

Source code :

```
model(REACT *react, int mode)
#else
model(react, mode)
REACT *react;
int mode;
#endif
   {

      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;

      switch(mode) {

            case USE_EB:
```

```
            {
            /*  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;
            break;
            }

            case USE_FR:
            {
            /*  Eb determination -------------------------*/
            z=R1/R;
            Fr=react->Fr;
            Fr1=Fr/(z/(2*alpha)*(cosh(delta)+alpha*sinh(delta))/sinh(delta*z));
            Fr2=Fr1/(R2/R1);
            z=R3/R2;

      Fr3=Fr2/(z/(2*alpha)*(cosh(delta3)+alpha*sinh(delta3))/sinh(delta3*z));
            react->Eb=Fr3/(Rb/PI/R3);
            break;
            }

            default:
            {
            display_error("Incorrect model call ...\n");
            display_error("*** Program terminated ***\n");
            exit(0);
            }
      }


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


/******  nitrate saturation  ********/
```

```
coefN=1;
/***********************************/
      react->rxa=Rmean*coefN*coeft;
      react->rn=yn*react->rxa;


      }
```

Module : ctrlspir.c **Name :** diff_var

Parameters : VARS *diff_var, int diff_time

Output : double

Description : This function returns the variable variation during time diff_time (in minutes)

Source code :

```
#ifndef ADERSA
double diff_var(VARS *diff_var, int diff_time)
#else
double diff_var(diff_var, diff_time)
VARS *diff_var;
int diff_time;
#endif
      {
      double dvar_dt;
      int nb_samp;

    nb_samp=ceil(diff_time*60/(TSAMP_MAIN * SAMP_SPIRU));
      dvar_dt=diff_var->val[diff_var->i]-diff_var->val[(diff_var->i
      -nb_samp)&NB_SAMP];
      return(dvar_dt);
      }
```

Module : ctrlspir.c **Name :** control_spiru

Parameters : ScreenViews active

Output : none

Description : This is the control routine for the spirulina compartment.

Source code :

```
int control_spiru(ScreenViews active_reactor)
      {
      FILE *fp;
      int retval = 0;
      char buffer[300], buffer2[300];
      double  Fr1, Fr2, delfr;                                  /*in W/m2 */
      double  prod_ref,prod1,prod2,prod_max,prod_min;           /*in mg/h */
      double  qe_max, qe_min;                                   /*in l/h */
      double  dil;                                              /*in h-1 */
      double  cxa_moy , nit_moy;                                /*in mg/l */
      REACT   react;

    sprintf (buffer, "Accessing spiruline data ...");
    _outtext (buffer);

    acq_vars_spirulina();

    #ifndef __Monitoring
    display_status("Control running ...");
    #endif

      if(!(next_pfc_sp--))
            {
            /* control PFC algorithm */

            /* biomass concentration */
            cxa_moy=average_var(&cxa,10);
            nit_moy=average_var(&nitrate,10);

            /* production calculation  */
            production.sp=cxa_moy*qe_real.value;

            /* reactor state */
            react.Cno3=nit_moy;
            react.temp=temperature.value;
            react.Cxa=cxa_moy;

            /* flow and production constraints*/
            qe_max=qe_nom.value*(1+DQ);
            qe_min=qe_nom.value*(1-DQ);
            prod_max=qe_max*CXA_MAX;
            prod_min=qe_min*CXA_MIN;
```

```
            /* feasible production setpoint calculation*/
            cons_prod_real.sp=max(prod_min,min(prod_max,cons_prod_nom.value));

            /* real flow setpoint  and corresponding dilution rate*/
            qe_real.sp=qe_nom.value;
            if(cons_prod_real.sp/CXA_MAX>qe_nom.value)
                  {qe_real.sp=min(qe_max,cons_prod_nom.value/CXA_MAX);}
            if(cons_prod_real.sp/CXA_MIN<qe_nom.value)
                  {qe_real.sp=max(qe_min,cons_prod_nom.value/CXA_MIN);}
            dil=qe_real.sp/VOLUME_TOTAL;

            /* reference trajectory */
            prod_ref=cons_prod_real.sp-pow(LAMBDA,NHC)*(cons_prod_real.sp-
production.sp);

            /*first scenario */
            Fr1=Fr.value;
            react.Fr = Fr1;
            prod1=predimod(react,dil,NHC);

            /* second scenario */
            delfr=DFR*signe(cons_prod_real.sp-production.sp);
            Fr2=Fr1+delfr;
            react.Fr = Fr2;
            prod2=predimod(react,dil,NHC);

            /* Fr calculation */
            Fr.sp=Fr.value+(prod_ref-prod1)/(prod2-prod1)*delfr;

            /* constraint on Fr */
            Fr.sp=max(FR_MIN,min(FR_MAX,Fr.sp));

            /* light setpoint sended to P100 controller */
            react.Fr=Fr.sp;
            model(&react,USE_FR);
            Eb.sp=react.Eb;

            /* pump setpoint sended to P100 controller */
            act_pompe.sp=qe_real.sp/cal_pump.value;

            /* model output calculation */
            prod_mod.sp = predimod(react,dil,1);

            next_pfc_sp=DT;
            retval = 1;
            }
      send_vars_spirulina();

    sprintf (buffer, " done\n");
    _outtext (buffer);

      return retval;
      }
```

Module : ctrlspir.c **Name :** average2_var

Parameters : VARS *diff_var, int diff_time

Output : double

Description : This routine returns the average ^2 during diff_time (in minutes).

Source code :

```
#ifndef ADERSA
double average2_var(VARS *diff_var, int diff_time)
#else
double average2_var(diff_var, diff_time)
VARS *diff_var;
int diff_time;
#endif
      {
      int j;
      int i_samp,nb_samp;
      double average;

      average=0;
    nb_samp=ceil(diff_time*60/(TSAMP_MAIN * SAMP_SPIRU));
      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);
      }
```

Module : ctrlspir.c **Name :** fill_struct_var

Parameters : VARS *fill_struct

Output : none

Description : This function fills the slope val[i] with the current value.

Source code :

```
#ifndef ADERSA
fill_struct_var(VARS *fill_struct)
#else
fill_struct_var(fill_struct)
VARS *fill_struct;
#endif

      {
      int jj;

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

Module : ctrlspir.c **Name :** fill_struct_cpt

Parameters : VARS *fill_struct, double delta

Output : none

Description : This function fills the slope val[i] with the current value and delta between each value.

Source code :

```
#ifndef ADERSA
fill_struct_cpt(VARS *fill_struct,double _delta)
#else
fill_struct_cpt(fill_struct,_delta)
VARS *fill_struct;
double _delta;
#endif

      {
      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;
            }
      }
```

Module : ctrlspir.c **Name :** slope_var

Parameters : VARS *slope_var, int diff_time

Output : double

Description : This function calculates the slope of variable by the least mean square method

Source code :

```
#ifndef ADERSA
double slope_var(VARS *slope_var,int diff_time)
#else
double slope_var(slope_var,diff_time)
VARS *slope_var;
int diff_time;
#endif
      {
      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_MAIN * SAMP_SPIRU));
      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);
      }
```

Module : ctrlspir.c **Name :** slope_cpt

Parameters : VARS *slope, int diff_time

Output : double

Description : This routine calculates the slope of counter by the least mean square method.

Source code :

```
#ifndef ADERSA
double slope_cpt(VARS *slope_cpt,int diff_time)
#else
double slope_cpt(slope_cpt,diff_time)
VARS *slope_cpt;
int diff_time;
#endif
      {
      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_MAIN * SAMP_SPIRU));
      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);
      }
```

Module : ctrlspir.c **Name :** signe

Parameters : double x

Output : int

Description : Returns -1 if x is negative or 1 if x is positive.

Source code :

```
#ifndef ADERSA
 signe(double x)
#else
 signe(x)
double x;
#endif
      {
      x=(x<0) ? -1 : 1;
      return(x);
      }
```

Module : ctrlspir.c **Name :** predimod

Parameters : REACT react, double dil, int horiz

Output : double

Description : Subroutine used for calculation of the vaules of the model variables during prediction horizon for each scenario.

Source code :

```
#ifndef ADERSA
double predimod(REACT react, double dil, int horiz)
#else
double predimod(react, dil, horiz)
REACT react;
double dil;
int horiz;
#endif
      {
/* react: current reactor model state    (Cxa,Fr,Cno3,temp)       */
/* dil  : dilution rate                  (in h-1)                 */
/* horiz: prediction horizon             (in number of DT)        */
/* prod : predicted production           (in mg/h)                */

      double v, prod;
      int k;

      v  = react.Cxa;      /* current biomass concentration */

      /* model integration (sampling period 1mn) */
      for(k=1;k<=horiz*DT;k++)
            {
            double Delta;

            react.Cxa=v;
            model(&react,USE_FR);
            Delta=1/60.0*(react.rxa-dil*v);
            v+=Delta;

            }

            prod=v*dil*VOLUME_TOTAL;
            return(prod);
}
```

Module : ctrlspir.c **Name :** diff_cpt

Parameters : VARS *diff_var, int diff_time

Output : double

Description : This function returns the delta count during the time diff_time (in minutes)

Source code :

```
#ifndef ADERSA
double diff_cpt(VARS *diff_var, int diff_time)
#else
double diff_cpt(diff_var, diff_time)
VARS *diff_var;
int diff_time;
#endif
      {
      int j;
      int i_samp,i_prev,nb_samp;
      double total_count;

      total_count=0;
    nb_samp=ceil(diff_time*60/(TSAMP_MAIN*SAMP_SPIRU));
      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);
      }
```

B.10 .- Module : Ctrlnitr

Module : ctrlnitr.c **Name :** control_nitrogen

Parameters : ScreenViews reactor

Output : int

Description : This function is going to contain - when available - the control code for the nitrifying compartment. It is ready to be expanded.

Source code :

```
int control_nitrogen(ScreenViews active_reactor)
      {
      int retval=0;

    sprintf (buffer, "Accessing nitrogen data ...");
    _outtext (buffer);

      acq_vars_nitrogen();

      if (next_pfc_nt-- == 0)
            {
            display_status("Apling model for nitrogen compartment.");
        wait_time(1);
        display_status(" ");
            next_pfc_nt = DT_NITROGEN;
            }
    sprintf (buffer, " done\n");
    _outtext (buffer);

      return retval;
      }
```

Module : ctrlnitr.c **Name :** send_vars_nitri

Parameters : none

Output : none

Description : This is the function whose task is send - through the network - the data to the MICONs. Now is not in use, yet.

Source code :

```
send_vars_nitri()
        {
        }
```

Module : ctrlnitr.c **Name :** acq_vars_nitrogen

Parameters : none

Output : none

Description : This function reads - through the network - the variables of the compartment.

Source code :

```
void acq_vars_nitrogen(void)
      {
      char bf[200];

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

    read_var(&AI0301);
    read_var(&AI0302);
    read_var(&AI0303);
    read_var(&AI0304);
    read_var(&AI0305);
    read_var(&AI0306);
    read_var(&AI0307);
    read_var(&AI0308);

    display_status(" ");
    }
```

Module : ctrlnitr.c **Name :** init_vars_nitrogen

Parameters : none

Output : none

Description : Its task is to set the name to all the variables that will be used in the nitrifying compartment.

Source code :

```
init_vars_nitrogen()
      {
      REACT   init_react;
      double  delta;
      int     jj;

      first_time_nitrogen = 1;          /* Set for the first filter entry */
      display_status("Initialisation of variables ...");

      /* Initialisation of filter parameters */


/*  TAG and COMMAND name initialisation     */

    sprintf(AI0301.name,"LOOP0303");
    sprintf(AI0302.name,"LOOP0307");
    sprintf(AI0303.name,"AI--0303");

    sprintf(AI0304.name,"AI--0304");
    sprintf(AI0305.name,"AI--0305");
    sprintf(AI0306.name,"LOOP0304");
    sprintf(AI0307.name,"LOOP0305");
    sprintf(AI0308.name,"LOOP0306");

/*  Variables initialisation   */

    acq_vars_nitrogen();

    next_pfc_nt=DT_NITROGEN;

      wait_time(1);

      display_status(" ");
    }
```

B.11 .- Module : Ctrlrhod

Module : ctrlrhod.c **Name :** control_rhodo

Parameters : ScreenViews reactor

Output : int

Description : This function is going to contain - when available - the control code for the rhodobacter compartment. It is ready to be expanded.

Source code :

```
int control_rhodo(ScreenViews active_reactor)
      {
      int retval=0;

    sprintf (buffer, "Accessing rhodobacter data ....");
    _outtext(buffer);
    wait_time(1);
    sprintf (buffer, " done\n");
    _outtext(buffer);
    wait_time(1);

      return retval;
      }
```

Module : ctrlrhod.c **Name :** send_vars_rhodo

Parameters : none

Output : none

Description : This is the function whose task is send - through the network - the data to the MICONs. Now is not in use, yet.

Source code :

```
send_vars_rhodo()
        {
        }
```

Module : ctrlrhod.c **Name :** acq_vars_rhodo

Parameters : none

Output : none

Description : This function reads - through the network - the variables of the compartment.

Source code :

```
acq_vars_rhodo()
        {
        }
```

B.12 .- Module : Ctrlliqu

Module : ctrlliqu.c **Name :** control_lique

Parameters : ScreenViews active_reactor

Output : int

Description : When available, the control process, will be placed in this function.
The output will be a bitmap to manage the disk storage.

Source code :

```
int control_lique(ScreenViews active_reactor)
      {
      int retval=0;

    sprintf (buffer, "Accessint lique data ...");
      _outtext (buffer);

    wait_time(1);

    sprintf (buffer, " done.\n");
    _outtext (buffer);

      return retval;
      }
```

Module : ctrlliqu.c **Name :** send_vars_lique

Parameters : none

Output : none

Description : This function, in future versions, will be the responsible of sending orders to the liquefying compartments.

Source code :

```
send_vars_lique()
       {
       }
```

Module : ctrlliqu.c **Name :** acq_vars_lique

Parameters : none

Output : none

Description : This function, in future versions, will have the task of take the values to the variables for the liquefying compartment.

Source code :

```
acq_vars_lique()
        {
        }
```

B.13 .- Relation function - module.

Module	Name	Parameters
alarms.c		
	acq_alarm	none
	alarm_lique	none
	alarm_nitri	none
	alarm_rhodo	none
	alarm_spiru	none
	init_alarms	none
	send_alarm	none
ctrlnitr.c		
	acq_vars_nitrogen	none
	control_nitrogen	ScreenViews reactor
	init_vars_nitrogen	none
	send_vars_nitr	none
ctrlrhod.c		
	acq_vars_rhodo	none
	control_rhodo	ScreenViews reactor
	send_vars_rhodo	none
ctrlspir.c		
	acq_vars_spirulina	none
	average_var	VARS *diff_var, int diff_tim
	average2_var	VARS *diff_var, int diff_tim
	control_spiru	ScreenViews active
	copy_react	REACT *reacta, REACT *reactb

Module	Name	Parameters
ctrlspir.c		
	diff_cpt	VARS *diff_var, int diff_tim
	diff_var	VARS *diff_var, int diff_tim
	fill_struct_cpt	VARS *fill_struct, double de
	fill_struct_var	VARS *fill_struct
	init_vars_spirulina	none
	model	REACT *react, int mode
	predimod	REACT react, double dil, int
	send_vars_spirulina	none
	signe	double x
	slope_cpt	VARS *slope, int diff_time
	slope_var	VARS *slope_var, int diff_ti
ctrlliqu.c		
	acq_vars_lique	none
	control_lique	ScreenViews active_reactor
	send_vars_lique	none
gpsfile.c		
	active_grp_gps	none
	close_grp_gps	none
	open_file_gps	none
Help.c		
	help	none
melfct.c		
	timebase_main	none
	wait_time	int seconds

Module	Name	Parameters
Melissa.c		
	alarms	none
	control_process	ScreenViews active_reactor
	copia	char *source, char *dest, un
	check_disk	none
	checkfloppy	none
	disk_storage	int input
	keybdrv	ScreenViews* react, char inp
	move_files	char *source, char *dest
	my_interrupt	none
	parameters_acquisition	none
	remove_file	char *name
	set_defaults	none
results.c		
	copy_disk	none
	directory	none
	error_log_file	char *input
	file_menu	none
	init_log_file	none
	init_output_file	FILE *fp, char *name
	log_file	char *input
	move_disk	none
	result	ScreenView *display, int new
	result_nitrogen	none

Module	Name	Parameters
results.c		
	result_principal	none
	result_reactor3	none
	result_reactor4	none
	result_spirulina	none
	result_status	none
	tech_log_file	char *input
Screen.c		
	AdvanceScale	none
	InputKey	char* buf, int maxim, int onlynumbers
	get_input_string	char *buffer,short x,short y, int maxim
	get_input_number	char *buffer,short x,short y, int maxim
	calculate_between	long begin, long end
	center_display_result	char *output, short y
	display_activ_group	GPS_FILE *gps
	display_no_activ_group	none
	display_region	char *error
	display_result	char *output, short x, short
	display_status	char *string
	display_time	char *time
	DrawAxes	int numpoints, long begin, l
	drawgraph	int new, int intervale
	InitializeGraphicalStructures	void
	DrawLegend	none

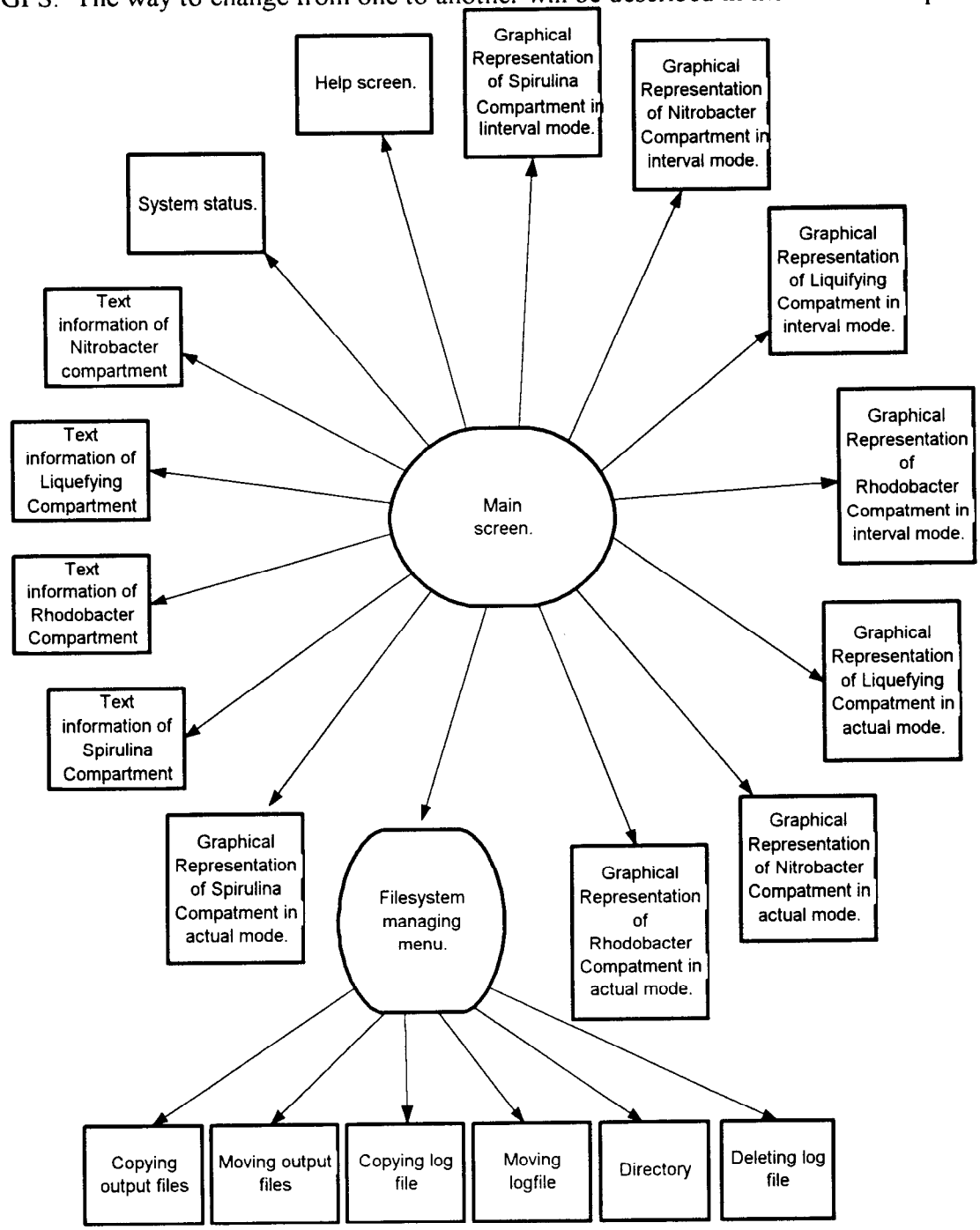
Module	Name	Parameters
Screen.c		
	DrawScale	int number
	Draw	int np, long begin, long end
	GoBackScale	none
	InitialitzeSpPlot	none
	next_control	none
	ReadDraw	char *name, int np
	screen_init	none
	SelectSpirulineGraph	none
	IncrementScale	none
	DecrementScale	none
	set_title	char *title
	tab	short x, short y
	testfileexist	char *name
	CalculateNextDate	long date
	use_control_window	none
	use_display_window	none
	use_group_window	none
	use_message_window	none
	use_status_window	none
	use_time_window	none
vars.c		
	check_network	none
	error_gps	none
	filter	VARS *var
	init_vars	none

Module	Name	Parameters
Vars.c		
	read_gps	VARS *read_var
	read_var	VARS *read_var
	selectcoeffient	char *name
	set_gps	GPS_FILE *gp
	write_gps	VARS *write_var
	write_vars	VARS *write_var

Addendum C .- Menus and control keys.

C.1 .- Menus.

This subchapter is going to describe the different screen and menus interrelations in the GPS. The way to change from one to another will be described in the next subchapter.



C.2 .- Control keys.

This subchapter contains a table that shows the function of each key depending on the screen that is currently selected.

Key	Screen	Function
Escape	All	Go to main screen.
F1	All	Display the help screen.
F2	All except the filesystem managing menu and derivated screens.	Shows the system status.
F3	All except the filesystem managing menu and derivated screens.	Go to the filesystem managing menu.
F5	Filesystem managing menu.	Selects the moving output files to a floppy disk.
F5	All	Shows the text information screen of the Spirulina Compartment.
ALT+F5	All	Shows the graphic representation of the Spirulina Compartment data in actual mode.
CTR+F5	All	Shows the graphic representation of the Spirulina Compartment data in interval mode.
F6	Filesystem managing menu.	Move the log file to a floppy disk.
F6	All	Shows the text information screen of the Nitrifying Compartment.
ALT+F6	All	Shows the graphic representation of the Nitrifying Compartment data in actual mode.
CTR+F6	All	Shows the graphic representation of the Nitrifying Compartment data in interval mode.
F7	Filesystem managing menu.	Remove the log file from the hard disk.

Key	Screen	Function
F7	All	Shows the text information screen of the Rhodobacter Compartment.
ALT+F7	All	Shows the graphic representation of the Rhodobacter Compartment data in actual mode.
CTR+F7	All	Shows the graphic representation of the Rhodobacter Compartment data in interval mode.
F8	Filesystem managing menu.	Displays the working directory, only are displayed those files whose extension is "out" or "txt".
F8	All	Shows the text information screen of the Liquefying Compartment.
ALT+F8	All	Shows the graphic representation of the Liquefying Compartment data in actual mode.
CTR+F8	All	Shows the graphic representation of the Liquefying Compartment data in interval mode.
F9	Filesystem managing menu.	Copies a file to a floppy disk, and the file to copy must have as extension "out".
+	Any graphical screen	Selects the scale of the next variable.
-	Any graphical screen	Selects the scale of the previous variable
*	Any graphical screen	Multiplies by two the actual scale values allowing a runtime scale adjustment.
/	Any graphical screen	Divides by two the actual scale values.
CTR+C	All	Exit the GPS program.