PROTEUS ES

Programming manual

Volume 1

Programming by means of graphical interface
Progamming in laguage C

The Graphical Interface

The interface of Proteus apparently has few objects of base, but thanks to the possibility to create and to use custom objects, and thanks to the methods and the properties of the existing components, studied for the maximum flexibility, it is possible with such instrument to create Forms and Applications of any kind and degree of complexity.

1– Objects and components

The graphical interface of Proteus allows to arrange on the FORM of the plan (the visualized pages) several objects, to which they are associated of the properties and the events. Proteus automatically assigns a name of default for the created objects. We can change to this name, changing the property nametype. The name of an object must be a alphanumeric letter string and numbers containing you do not space and must begin with a small letter. In the name is concurred to use the special character "_" (underlinig).

An object makes part of the form in which it is defined.
The properties of an object they make part of the same object.
As an example:

If in page page1 we have the object label1:

- label1 it is known from the program like page1-> label1
- its property caption is `page1->label1->caption.`

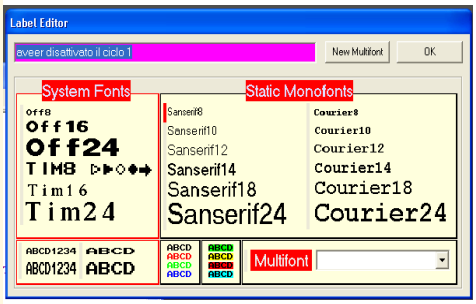
In order to favor the possibilità to carry of the code, that is in order to write of the routines that they can be used also on various objects or various pages, we can use the following names **of default**:
Inside of the page (as an example in the events of the same object or other objects of the page), the same page is possible' to be called like **Local.**
Inside of the events of the object, the same object is possible to be identified like **me or This.**

Example:
In the code associated to the events of the objects of page1,
`page1->label1->caption` it is equivalent `Local->label1->caption`
In the code associated to the events of label1,
`page1->label1->caption` it is equivalent `This->caption`
Or to `me->captino`

The predefined objects of base

Label:

it allows to visualize writing on the form. The visualized written is the property caption of the object.
The properties **font, fontcolor and color** allow to change the font respective, the color and the color of the bottom of the written one



With a double click on a label a window is opened in which it is possible to change the main ones proprietá (font, color, fontcolor, caption) in simple and intuitive way.
The Caption property deserves a speech more investigated.
If Caption contains a simple text, this text will come visualized, or translate in the running language in the event in which it is anabled the option multilanguage (you see option multilanguage).

In Caption alternative it can contain the name of a addres.

As an example:

caption	*pippo
caption	msg[page+7]

In such case the written one will be assigned dynamically.

In order to change the caption from program, we can use more ways:

```
1- page1->label1->caption="abcd";
2- sprintf(page1->label1->caption,"prova%d",varx);
3- strcpy(page1->label1->caption,page3->label4->caption);
```

In the first example we change the addres, while in the following ones we rewrite the content of the string. In the second case we must make attention not to write a long string more of that originates them (that defined in the property caption) otherwise we do not know what we go to overwriter in memory.

.

Edit:

It allows to visualize the associated value of a variable one (property **var**). The property **caption** allows to add an eventual one written to right of the same field..

With the exception of the Label object, a field of Edit allows to visualize variable alphanumeric (strings, that is char *) and numerical, specifying the number of characters (**digits**), the position of an eventual point decimates them (**decimals**), the style of the frame (**bevel, bordercolor, borderwidth**).

If the property **enabled** is worth 1, the operator can change the value of the variable one associated from keyboard or with mouse or the touch-screen. In any case, the visualized value extension always the present value them of the variable one. If we want to increase the time of variable refresh (the useful one for variable that they change a lot quickly), we can introduce in the property **frequency** the minimal time (in milliseconds) of refresh of the field.

The properties rows Y and columns X allow to define with a single object a vector or a matrix of fields of edit. In such case it is possible to assign to the property var (variable associated) a vector or a matrix, using the intrinsic indexing [].

As an example if for edit1 we define:

We will have 4 you live of edit aligned vertically, the first associate to pippo [0], the second to pippo [1] etc….

var	pippo[]
Rows Y	4
Columns X	1

We will have a matrix of two columns and 4 lines of fields of edit, associated to caio [0] [0], caio [0] [1] etc….

var	caio[][]
Rows Y	4
Columns X	2

Also for the Edit object, regarding the caption it is worth said how much for the label, and in case of matrices of fields of edit, we will be able to use for the caption, intrinsic matrices of strings.

as esempio:

```
char  scritta[][]={“line1”,”line2”,”line3”,”line4”};
```

caption	scritta[]
Rows Y	4
Columns X	1

Button:

it is the classic button of Windows. The variable one associated (property **var**), of type byte, will have value 1 when the button is pressed, and 0 when the button is not pressed. The properties **caption and font** concur to manage the written one on the button.

If we want a keyboard with more buttons mutually exclusive rights,

all the buttons must have the **same variable** (of type byte, that is char).
- the buttons must have the property **refresh=1**
- the buttons must have the various properties **tag.**

In such a way, pressing a button, the variable one assumes the value of the correspondent property tag, and the only button that will turn out pressed is that one with own equal property tag to the value of the variable one.

For having more independent keyboards, enough to use variable var various.

With a double click on the button, like for the label, a window is opened that concurs to set up in comprehensible way the style and the color of the written one.

The written one is of centered norm, but for some font monospaced cannot turn out a put off center . In such case it is possible to align inserting it at the beginning or at the end of the same one of the characters blank

For a corrected animation, the property bevel must be set rised2 (default) or rised1.

Checkbox:

It is equivalent to the checkbox and the radiobutton of Windows.

The variable one associated (property **var**), of type byte, will have value 1 when the checkbox it is active, and 0 when it is disable. The properties **caption and font** concur to manage the right written one of the checkbox same.

If we mutually want a series of checkbox exclusive rights (radiobuttons),

- all the checkbox must have the **same variable** (of type byte, that is char).
- the checkbox they must have the property **refresh=1**
- the checkbox they must have the various properties **tag**.

In such a way, pressing a checkbox, the variable one assumes the value of the correspondent property tag, and the only one checkbox that it will turn out active is that one with own equal property tag to the value of the variable one.
For having more independent series than checkbox, enough to use variable (properties var) various.

Panel:

It can be used like simple decorative element, in order to divide the page with raised, lowered or bordered frames (property **bevel**), or like active container. In fact, besides the pure decorative function, a Panel has the following operating functions:

The objects that come created inside of a Panel, they make part of the same panel, and therefore moving the panel, we will move also the objects that are found on it.

- If a panel changes own state from visible not visible (property **visible**), also all the objects in contained it are not visible.

For a panel, the property caption, var, refreshvartype, glyph, numglyph, frequency, although present, they do not have some sense in itself. The property enabled it renders the panel to the touch-screen and the mouse sensitive if the value is 1, activating the events on_click, on_mousedown, on_mousepress and on_mouseup. The property **canfocus**, if to 1, it renders the selectable panel from keyboard with the keys arrow-right and arrow-left, and qualifies eventual fields of Edit in contained it to the selection by means of the keys arrow-up and arrow-down. It is possible, for equipment with single keyboard, an access line-column to the fields of edit, in which the panel it represents the column.

Also for the Panel objects, as for the fields of edit, it is possible to set property **rows Y** and **column X** in order to create vectors or matrices of panels.

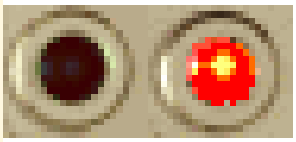
Bitmap:

It is the object that concurs to visualize designs, also animated.
The proprietá bitmap it is the name of the design to visualize. Such property one assigns in Proteus, choosing the figure (of type BMP) and regulating of the shade of colour, the dimensions and the appropriate transparency with tool the present one on Proteus and that it comes called automatically with a double click of the mouse on the ' object bitmap.

If the design is animated, that is if it consists of more figures that can change, it is necessary to leave from an image of type BMP in which present, **all of the same width in pixels** are placed side by side horizontally and, all the alternative designs. To this point we will put in property **numglyphs** the number of all image and in the property glyph what we want to see.

During the execution of the program the visualized design will be that one characterized from the associated value of the variable one **var (glyph)** (to value 0 corresponds the first design, that is that one on the left).

If for the property glyph we use a greater value of the number of designs (greater of **numglyphs** - 1)the bitmap will be not visualized in the window of edit and it will be showed the name of the bitmap same.
Also for the Bitmapl objects, as for the fields of edit, it is possible to set property **rows Y** and **column X** in order to create carriers or matrices of images.



As an example:
We have a design that represents led in its two states extinguish-ignited. We create a bitmap bitmap1 associated to this design, in which we put:

var	ingr[]
Rows Y	1
Columns X	8
Numglyphs	2
Bitmap	AALED2

In this way we create a row of 8 led. If we want to see on these led the state of the first 8 inputs of the cn400, we can as an example write:

```
char ingr[8];
void Event(bitmap1_onentry)()
{
   char x;
   for (x = 0;x < 8;x++)
      ingr[x] = I(i+1);
}
```

Object:

it is the **component** object, that is a complex object previously constructed and that it is found in bookcase of the objects. A figure is constructed using the objects of base over showed, and eventually others object.
The modularity is a point of force of Proteus, this allows the programmer to use objects precast concrete slabs also much sophisticated (PID, Posizionatori, Lapping machines etc.) and it moreover concurs to construct own objects by themselves, assembling or modifying of previously created or chosen others between those standard of Proteus.

To every object a name is assigned that coincides with that one of own proprietá **var**. Such name must compulsorily have the first very small letter. A object it is identified from own name, and in phase of drawing up of the program it is possible to decide itself if the objects are value absolute or, like all the other objects, make part of the page contains that them. In this last case to select checkbox the local objects. An absolute object does not have need of being recalled with page-> object but for against, cannot appear on various pages absolute objects with the same name.

An object, besides the name-variable one does not have other properties and events, but it inherits those of the bookcase object originates them. Like such, it can have associate many variable-properties, according to its complexity.
- All the objects have the property val, that it coincides with the name-variable one of the same object.
- The properties belong to the bookcase model, that it has the same name of the object, but begins with it them CAPITAL.

We make an example.

We take a switch of Switch8-40 type, that we will call interrutt1 (we write interrutt1 in the property var and write what we want in caption).

With the instruction `interrutt1=1` the switch goes on (and will see the image with the raised –swtch).

The same thing if we make `Interrutt1->val=1.` (to notice the capital letter).

In the part dedicated to the objets standard, for every object is listed the variable ones associated.Regarding the objects created from the programmer, it will be task of this last that one to remember the name of the variable ones associated (those declaring **friend** in the form of creation of the object). We will more ahead see in the details the operation of creation and modification of an object

Static Label:

it is equivalent to the Label object, but it is an object fixed that does not allow to change the caption from program.

Static Bitmap:

it is equivalent to the Bitmap object but it is a fixed object, that it contains `an only static image.

These types are present for historical reasons of compatibility with versions precedence, but they do not have to be used for new plans, as they will not be more present in the next versions of Proteus.

Multipanel:

it is an object that serves in order to create the base for files. We see with a practical example the creation I use and it of a multipanel:

- We select with mouse the left on the bar of the menu the object multipanel. - Click on the form in order to place the multipanel where we want. We notice that checkbox the multipanel has been selected automatically.
- With the mouse skillful we reorganize the panel and with to that we move it left where wished.
- To this point we choose on the bar of the menu the object panel we place and it within the multipanel as soon as created. We will see that panel assumes dimensions of multipanel, that two diagonals appear in order to say that the panel is not enable, and will see the written Pag. 1 up on the left. with to that we move it left where wished.
- We repeat the same operation others two times, creating 2 Pag and Pag3.
- To this point we select on the bar of the instruments the Button object, and place it within the multipanel. We will see that the button is aligned up on the left and assumes caption Pag 1.
- We repeat the thing creating the buttons 2 Pag and Pag 3.

We exit checkbox the multipanel now. In this way we have the fire on the single pages and not more on the total multipage. The diagonal bars are disabilitate and will be able to change page with the buttons up, or with a double click of the mouse inside of the same page.

- Every page is to all the effects a panel, and to a data moment the objects will be visible only that we have created on the selected page of the multipanel.
- It is not necessary to create the selection buttons, but if we do not make it we will be able to change to page only from program but not manually (changing the property glyph of the multipanel) , not being the buttons in order to make it to us.
- A time created the selection buttons, we can change of to the name rewriting the proprietá caption.

The objects till now seen find on the bar of the **objets standard**.The bar of the objects Applications contains other objects (Applications, Page, Timer Hardware, Serial Port etc…) that we will see after. With the exception of the objects standard these last ones are always active, even if they are not on the page in which they have been defined. The only interesting menu member such, for the time being is

Timer:

It is an object that has the only properties enabled and interval and the method on_timer.

If the property enabled is true (various from 0) the event on_timer it is executed every “interval” milliseconds.

The properties interval and enabled can be changed from the program at any moment.

The properties

Every object has of the properties, of the variable determines its aspect and if changed from the program modify the same object.
The variable ones in issue can be of the char, the word, the strings or other. Every property has its type, or its types, of variable.
The proprietá they can be changed from the inside of the same object, from routines (or methods) of other objects of the page, from routines in other pages, the cycle of PLC…. In other words, any part of the code can be change the property of any defined object (you see later on: definition and visibility of the objects).
For this purpose the properties they must be globals variable. In order to make this all the properties of the objects are contained in a structure associated to the same object. All the structures of the objects of a page are aimed from an ulterior structure associated to the page. Therefore a property is determined by means of the development of the program:

Page->object-> property

Only for the objects of component type, the option to render the total objects is selectable (not legacies to the page). If such option is activated, the property he is attainable by means of the route:

Object->demo demo

We now examine in the details the meaning of the properties of the objects.
We hold account that all objects do not have all properties (as an example a Label do not have Bitmap neither text frame), but that the properties present have the same meaning of the all objects (to es. the property Top have the same meaning , independently from the fact that features of a bitmap, a label or a button).

nametype

It is the name of the object. In effects it is not a true one property but it is the name of the structure that encloses all the others properties of the object. It is possible to use the same one nametype for various objects in various pages. If instead we use the same one nametype for various objects in the same page, the successive objects to the first one will be of clone of the first object.
An object clone works very well, and generally it comes used this method in order not to uselessly multiply the objects of a page and the relative code, in the event of similar objects. As an example carriers of panel and of fields of edit are made by means of clonazione.
The objects clone have however some limitations:
The events come inherited from the first object,that it is the only one in which the relative fields they exist.
The properties fundamental they exist and they are distinguished for every object clone. The properties fundamental they are:

Caption
Var
Left
Top
Tag

Such properties they are however static and they cannot be modified from program not even for the object father.

All the other properties (width, height, color…) they are static, cannot be modified, they are visible and setting only on the object father, and are equal for all the which cloned objects.
Nametype is a label, and therefore it must obey to the rules of language c for the names of the label, that is must be a letter series and numbers, the first character must be a letter, is admitted the single underscore special character (_), cannot have the name of words key of the c (for, if, sin, cos, printf etc)

caption

It is a string associated to the object.
It can represent the text (label buttons, checkbox) or a comment (edit). If it is a fixed string it must contain more of 64 characters and it comes managed from the tool of automatic translation, in the event in which in the plan a greater number of languages of 1 is declared.
In effects the automatic translation is “automatic” only in the sense that, in the moment in which the varied value of the variable Lang, all caption of all the visible objects change and show the text translate in the new language. Some universal dictionary of translation does not exist however, and in effects such dictionary it we must make, in the following way.
After to have compiled the program, in the directory of the same program we find rows that LANG2.H is called. In it they are found, repeated n times, where n it is the number of languages indicated in the plan, all caption of all the objects of the plan.
In the moment of the creation, all the strings are equal to that present in the field caption. It will be our cure cure to edit such file with a editor of text and translate the several recurrences of the strings in the languages that we want.
When the variable Lang is worth 1, it will be looked the first string, with 2 second the etc.

As an example, if number lingue=2 and we have :

nametype	label1
caption	Prova

We will find after the compilation, in LANG2.H, a line

```
const char *L_Prova[]={"Prova","Prova"};
```

That we will translate as an example

```
const char *L_Prova[]={"Prova","Test"};
```

Obviously, translate we must make attention to write of the strings that are not too much long and they do not go to goverflow from the fields. In principle not there is some limitation to the number of characters of a translate string. Wanting, we can also modify before tightens (that in the language originates them).

An other way to inizializzare the variable one caption is that one to write in the same field the name of an address, as an example:

```
*string1
string1[0]
String1[]
String1[][]
```

The third form (implicit index) is usable for vectors and matrices of objects (only sees fields of edit and label) the quarter forms (implicit index double) is usable only for matrices of objects.
In the event of objects of component type, the property caption comes of used usual in order to pass an optional parameter to the object, and its meaning changes according to the object.

var, vartype

The property **var** is the main difference between Proteus and the other RAD, like Borland Builder, Visual Basic, Delphi etc
In Proteus every object is variable legacy dynamically to a total one of the plan. If the variable one changes, the object reacts to such change (to es. bitmap an extension `a various image, a field of edit show a various value). Moreover if we carry out determined actions on the object (to es. if we press a push-button) the variable one associated it changes.

Var is the variable name of a total one (that it must be defined from programmer) the
Vartype it is the such type of variable, and can be chosen from menu between the following types:

Byte (char)
Word (short int)
Doubleword (long int)
Real (float)
String (char*)

In vartype we must indicate the type with which it has been defined the variable one. If we mistake ourselves (to es. we indicate like byte a variable one short int), the program can give turned out uncontrolled or stop.
In var we can put also an expression, provided that its form is compatible with an expression that is to the left of the equal sign.
As an example

```
Quotax[testa1*para2+3]
Quotax[quotay[zz2]]
```

They are correct forms while

```
Quota1 + 3
```

It does not go well, because

```
Quota1 +3 = 0
```

it is one error in C .

For vectors and the matrices of objects it is possible, as for the property caption, to use the implicit form to index

Var[] e
Var[][]

In case variable the changes a lot quickly (as an example the quote during a positioning), can be placed of the limits to the frequency of refresh of the variable one on display, setting the property frequency to a enough high value (are the minimal time of refresh of the object in milliseconds).
Every object can have the property frequency various from the others. At the moment of the creation to the object comes associated the variable VAR1 of default. This is a variable one of system allready defined and can be left if the property var it does not interest (for es. for the label).

It is variable an only one for every object, and is that which, with to tag, distinguishes an object clone from an other .
.

Tag

It is a number, that it can be used for various scopes, at the discretion of the programmer.

Generally it is used for similar objects (to keyboards, vectors of checkbox etc) in order to characterize of the number.

For buttons and checkbox mutually exclusive rights, it contains the number to assign variable the associate one when the object is active. For objects clones, with to the property var, is the only way in order to distinguish a clone from an other.

It must be defined with an integer in the range 0-65535. It cannot be variable neither an other.

visible enabled

Visible, if true (various from 0) renders the object visible, cancels otherwise it from the display.

Enabled, if true, enabled the sensitive object to the touch-screen (or to the click of the mouse).

These properties, with the exception of all the others, are addres to the respective property, therefore the program to aacces must use the differentiation.

As an example:

```
*page1->edit1->visible=0;
*page2->button3->enabled=1;
```

To notice the asterisk in front of the name of the property.

A not qualified object is active and regularly works (as an example a field of edit visualizes the changes of the variable one associated), but he is not sensitive to the touch.

frequency

This property assigns the minimal time of refresh of an object.
The objects come dawned (redesigned) when it changes a property that involves a variation of the representation of the same object. In sure cases besides the object it must be redesigned also what it is around (as an example if an object comes moved, is necessary to redesign the background in order to cancel the old object).

All this demands of the time, and if or more objects vary very quickly (we think next to the quote an axis during the positioning) the continuous one refresh can provoke to slowing-down of the interface customer (processes PLC do not suffer any as they turn in an other task), flickering and annoying visual effects. In order to avoid this, we can assign to such objects a higher time of visualization (10-20 is a good compromise). Or, in the event in which or previewed that they change of the properties that they demand to redesign the object, it is well to include it within a not transparent panel. In such a way alone the panel will be redesigned and not all the form.

besides the value of the variable associate, the properties that, in variation case they demand to redesign the object are:

Borderstyle (bevel)
BorderColor
Color
FontColor
Glyph
Tag

The properties that in variation case they demand to redesign also the background are:

BorderStyle (if with the new style the frame is thinner)
Border
Caption
DecimalPointPosition (decimals)
NumDigits (digits)
Height
Left
Top
Width

Specific properties of the Edit object

The fields of edit have some dedicated properties.

Font: it is the font of the field of edit and also of the string caption.
Color: it is the color of bottom of the window of edit.
Bordercolor: it is the color of the edge of the window of edit and the caption.
Borderwidth: it is the extra-size (in pixel) of the background of the window of edit.
Fontcolor: it is the color of the written one in the field of edit.
Rows Y: if > 1 defines a vector vertical of fields of edit.
Columns X: if > 1 defines a vector horizontal of fields of edit. (if or Rows that Columns is > 1 is a matrix of fields of edit)
Bevel: it is the style of the edge of the field of edit (in relief, lowered etc)
Decimals: it defines the number of decimates them after the comma. If the type of variable is entire, it defines the fictitious position of the point decimates them. As an example: if the variable one is a Word and is worth 1000 and decimals=2, value 10,00 will show on the field , if we write new value 3 will visualize number 3,00 and the variable one assumes value 300).
Digits: the number of figures or letters is the fixed width of the field (that can contain, comprised the sign and the point decimates them.

Specific properties of the Bitmap object

The type fields bitmap have some dedicated properties.

Color: it is the color of bottom of the field (visible in the transparent zones of the image)
Bordercolor: it is the color of the edge of the design.
Borderwidth: it is the extra-size (in pixel) of the background of the design.
Rows Y: if > 1 defines a vertical carrier of fields of edit.
Columns X: if > 1 defines a horizontal carrier of fields of edit. (if or Rows that Columns is > 1 is a matrix of fields of edit)
Bevel: it is the style of the edge of the field of edit (in relief, lowered etc)
Bitmap: it is the name of the image to visualize.
Numglyphs: it indicates the number of under-images in which the image to visualize is uniform horizontally.
Select: it indicates the number of under image visualizing.
If Numglyphs is > 1, glyph automatically assumes in the program the associated value of the variable one. Such field is present in Proteus only for a preview of the graphical effects during the drawing up of the program.
Scalable: if place to 1 allows to vary the dimensions of the object, concurring some zoom or a reduction with pleasure. Selecting the case fix zoom they will be only concurred details values for zoom (50%, 75%, 100%, 150%, 200%). The zoom it is always the same one for the two aces, and it does not concur distortions of the image. **The Scalable property exists also for the objects of Component type, but in such case he is not modifiable, that is: if a member is constructed in order to be varied, its name must end with the suffix _r, and the variable property will be activated automatically.**

Other properties :

The properties **width, height, top, left** inside give to the dimensions and the position of an object (of the page, or the panel contains that it)

The properties row, column, canfocus, canedit are used for fields of edit, and used above all with interface keyboard without touch screen.
Row is the verticdemo demo demo demo demo
Column is the horizontal position of the field.
By means of the keys arrow it can be mouving between the fields of a page.
The accessible fields are those of the objects of Edit type that:
They are contained in a visible panel
They have the various properties row and column from zero
They have the demo demo demo demo
The editabili fields are those which, besides the conditions over sights, have the property canedit=1

The property refresh, that it can be worth 0 or 1, have various meanings, according to the type of object:
Edit: if refresh it is 1 field will come refreshed even if the value of its variable one does not change.
- ***Button***, ***Checkbox*** : if refresh it is worth the 1 object it is mutually exclusive with those which they have the same one var, and is active when the value of the var is equal to own tag.

Fontx and **Fonty** come used, in coupling to the Font property and limitedly to font with zoom (the OFFx and TIMx) for the zoom horizontal and vertical of the font same

Align it is used practically only for the panels, and serves to align them to the page or the panel contains that them.

Mobile is only used for the panels and concurs to render them moving, that is movable from a place to the other of the Form by means of the touch screen

Practical rules for the corrected use of the objects

The Proteus system is much flexible one and allows to create the graphical interface in interactive way and with little it touches of the mouse, however if some practical rules are not observed, the result can be poor and to give rise to flicker and slowin-down in the visualization.

- We must keep in mind who when in the program we change the properties of the objects, if this carry to a physical change of the aspect of the object (visibile-no visible, reorganization, movement, change of the text frame, change of the caption or the color etc.) the object will come immediately redesigned.
In more the overlapped objects to it will come redesigned all.
In the event in which the simple one to redesign the object he is not sufficient (as an example if we rewrite writing with transparent bottom, if we make smaller a panel, if we change to a design) the graphical motor can think necessary to redesign the bottom in order to cancel the part of the object not more visible. This involves that the objects overlapped to the redesigned bottom will come redesigned also all.
Like consequence, if as an example we change the caption of a label, all the page would come redesigned, that, to part the slowing down, provokes an annoying visual effect.
In order to obviate to this, it is well to place the objects that can be redesigned inside of a panel of not transparent color, so as to limit the area to redesign
It is not necessary to take such precaution for the objects when they come redesigned automatically, as for the fields of edit when it changes the visualized value, the buttons when they come pressed, the checkbox when they come selected or disable, the multipanel when we change panel
If we use of the bitmap that they can change, if these have of the transparent zones, is well to define the bottom of the bitmap with a not transparent color, otherwise when changes the image, the old image in the transparent zones of the new image could remain in evidence.
It is well moreover not to create animated images much large, as the refresh he would be slow and visible. In such case it is better to create a fixed great image, and in overlapped of the small images animated for the details that they change. These norms are not compulsory, but dictated from the good sense. The experience of the programmer can suggest other astuteness in order to render more fluid the program.

The events

To every object they are allottable of the events, that is the functions defined from the programmer, that they come executed when they are taken place determined conditions.
The events are functions without parameters, however when calls come, there is a sure number of variable total that assume determined values:

MouseX (o **mouse_x**) horizontal position of the mouse (or the touch on the screen) relative to the same object (0,0=angolo up on the left).
MouseY (o **mouse_y**) vertical position of the mouse (or touch on the screen) relative to the object the same (0,0= angle up on the left).
This (**me**) addres to the same object.
Local addres to the same page .
Var associated structure to the variable one.
-**Var.Byte** (**_byte**variable associated (case char,unsigned char)
-**Var.Word** (**_word**) variable associated (case short int)
-**Var.Dword** (**_dword**) variable associated (case long int)
-**Var.Real** (**_real**) variable associated (case float)
-**Var.String** (**_string**) variable associated (case char [ ]).

As an example, if inside of the event onentry of an object we write:

```
if (_word > 10) _word=10;
```

We limit to the maximum rating 10 the associated value of the variable one.
In the code of an event we can moreover read and modify the properties of the object same (This-> property), of the other objects of the page (Local-> object-> property) and of all the other objects of the plan (page-> object-> property).
If the code is written using the variable ones of default (This, Local, _byte, _word etc.) also other objects of the page can use the same event, and the code does not demand some modification if we change the name of the page or we copy if it in order to use it on an object of an other page.
It is necessary to make attention, referring to other objects, than these last ones they exist, that is that already they have been created. It is good norm to use a code of the type:

```
if (page1->label1) page1->label1->caption="abcd";
```

We now see the meaning of the events of the objects.

on_first

This event comes called for every object in the moment in which a new page is opened.
We can as an example write the initialization code.

on_create

This event comes called every time that the entire page comes redesigned. Except the case in which forces from program refresh of the page or a decoy from a form child, it is equivalent to the property on_first.

on_entry

This event comes always called, before any other event (except on_first and on_create) when the every millisecond is made polling of the object (in average). In this subroutine we can as an example put the code of control of the parameters of the object, of the visibility and the qualification of the same one.

on_draw

It comes called, in the event in which an object it must be designed or redesigned, before designing the same object. It is here that ulterior controls can be made and eventually to design various and to cancel the action standard of design (**putting to zero the variable ActionEnable** that is the flag of qualification to the action, that is the design)
In the event of objects of Panel type, if present the event on_draw, it will have to be occupied also of the eventual design of the bottom of the panel (if not transparent) as, only for the objects of Panel type, the event on_draw **replaces the action of default**, which will not be executed.
This uses concurs it of a panel like associated active window to subroutines details.

on_click

It comes called when the object comes touched on the touch-screen, or when you click with the mouse, or when the object is evidenced and a key is struck on keyboard. Also in this case controls can be made and eventually to cancel the action of default **putting to zero the variable ActionEnable**

on_mousedown

It comes called a single time in the moment in which click with the mouse on the object or it is touched on the touch-screen.

on_mousepress

It comes the entire time called in which mouse (or the touch-screen) it remains pressed on the object

on_mouseup

It comes called a single time to the release of the mouse (or the touch-screen).

on_exit

As on_entry it comes at the beginning called of the polling of an object, thus on_exit it is called at the end of the polling same.
It can be used in order to complete other actions precedence. As an example, in the library of the objects standard, for the objects of the type analogic instrument, in this event it comes redesigned the pointer of the instrument if the instrument has been redesigned or if the associated value of the variable one is changed.
To such scope, it comes conserved a trace of the calls of the events of the object in the variable ones:

_abilita_azione_click =1 if executed on_click
_abilita_azione_repaint =1 if executed on_repaint (*)
_abilita_azione_draw =1 if executed on_draw

(*) the action on_repaint is an inner subroutine and comes called when the object comes completely redesigned. As an example for the field of Edit on_repaint it redesigns all (frame, background, caption), while on_draw write again only the value of the data. Being on_repaint subordinated to the management of the diagram and not to external actions, he is not expectable and it does not exist a usable corresponding event from the programmer. The only place in order to make something in this case is in on_exit using the flag _ qualifies _ action _ repaint).

2 - The structure of the project

A project-type of Proteus consists more files, all contents in the directory of the project, of which we will see the meaning. **With Proteus a directory it can contain a single project.**

The pages:
For default the files of the pages calls come page1.c, page2.c etc. But nobody prohibits to assign they an other name to the moment of the creation.
The first page comes created automatically when a new project is created. The successive pages can be added from the menu the Archives-> New Page.

As soon as created the text of the page it contains includes only it of the header and the declaration of the form. We can add the routines to you for the events and declare variable (the total ones) of the page.
The variable ones must be declared include before it of the header, while the body of the routines associated to the events comes **created automatically** with a double click on the field of the chosen event, in the file of the inspector of the objects.

Exampe:

```
unsigned char pippo;  //   global variable
#include "page1.h"
// ------------------------------+
//    form  page1
// ------------------------------+
void form_page1(void)
{
  object(TPage,&page1,page1_Body,0,0,&Null,Byte,1,0,"PAGE");
}
// -------------------------------
// methods:
// -------------------------------
void Event(edit1_onentry)()
{
//  to write own code here
}
```

Effectively the testuale part of the page, written in language standard C contains only the declaration of the variable ones used and the relative code to the events that we want to deal in way not standard.

In the majority of the cases it is not necessary to write nothing, leaving the management of default of the objects.
The real construction of the page, whose code C comes generated automatically in the file header (pagexx.h) is executed in interactive graphical way using the graphical window, the bar of the instruments and the inspector of the objects.

The construction of allows us to the pages to implement the interface man-machine. Besides this, in a normal program of automation, the operating part is necessary, that it can be written in language C o/e in Ladder language.

The first compilation of a project, that it contains at least a page with at least an object over, it creates automatically the trace of all remaining files that we can edit, and that we go to examine. The files below they are edit in the text window, and they are opened by click with the mouse rigth inside of it, choosing the menu Open Files.

COMMON.H

In this file we must declare globals variable that they will be used from the several pages and on job program . Also at the beginning of the page, as we have seen, we can declare of the globals variable , but only if they only come used from the same page. In common.h we will write also the declaration of eventual structures, union, declarations of #define, functions and subroutines that we will use in the events of the pages and into cycle machine.

COMMONPLC.H

Here we will declare variable, the structures and the functions that will not be used from the events of the objects, but that they use like parameters property and/or events of the objects, and that therefore they must be defined after the definition of the pages (exactly in commonplc.h).

START.C

It contains the initialization code, that is all the one which we must make a single time to the start of the program (loading parameters, initialization of the com, the encoders, the quotas of the axis etc…) The variable ones used must be declared in common.h or commonplc.h.

PLC.C

It is the part of the code of automation of the machine that is written in language C.
As soon as created it is a file of the type:

```
//----------------------------------------
// PLC CODE program Base
// to insert here the code of the PLC
//----------------------------------------
if (first_cycle)  //INITIALIZATION//
{
//----------------------------------------

//----------------------------------------
}
if(is_running)    //CICLE PLC//
{
//----------------------------------------

//----------------------------------------
}
idle();
//----------------------------------------
```

The first condition comprises the code to execute to the start, second the code of the machine to regime. It does not have to contain loop of waited or routines sluices on same himself.
It can recall functions and use variable declared in common.h.

LANG2.H

It contains the dictionary of the phrases translate if the support has been activated multilanguage. That one translate the successive occurrences to before will be our task in the languages that we want to use.

3–The functions of the library .

Proteus is born mainly like tool of programming for equipment with PLC, therefore it is equipped with base with a series of functions in a position to managing the devices of the PLC, which

Kernel
Encoders
Timers
Clock realtime
Derivate
Delayed
One shot
Graphical functions
Keyboard
Serials port
Flash SD/MMC
Pen usb
Filesystem
Can
Canopen
Ethernet
Expansions i-o

We now examine the functions and the meaning of the relative parameters

KERNEL

```
int  exec_task(void(*fun)(),int numtask,int priority);
int  resume_task(int numtask);
int  suspend_task(int numtask);
int  remove_task(int numtask);
void _idle(void);
int  _p(int event);
int  _v(int event);
void _waitfor(int event);
void _event(int event);
```

ENCODERS

```
void load_asse1_fast(volatile long int *quote);
void load_asse2_fast(volatile long int *quote);
void load_asse3_fast(volatile long int *quote);
void load_asse4_fast(volatile long int *quote);
void load_asse1(void(*)(),void(*)());
void load_asse2(void(*)(),void(*)());
void load_asse3(void(*)(),void(*)());
void load_asse4(void(*)(),void(*)());
void find_zero(unsigned char n,long int quota);
void clear_asse(unsigned char n);
```

INTERRUPTS

```
void load_interrupt(int num,void(*)());
```

ANALOG INPUTS

```
void set_anal(int num,int level);
void refresh_anal(int num);
void pot_init(void);
void pot_start(void);
void pot_stop(void);
void pot_refresh(void);
```

DIGITAL IN OUT

```
unsigned char I(unsigned int n);
unsigned char O(unsigned int n);
void out(unsigned int n,unsigned char x);
void __i(void);
void __o(void);
```

TIMERS

```
int  wait(time_t n);
int  every(time_t n);
void time_init(void);
int  exec_timer(void(*subroutine)(),time_t _interval,time_t fra_quanto);
int  resume_timer(void(*subroutine)());
int  suspend_timer(void(*subroutine)());
int  remove_timer(void(*subroutine)());
int  sleep_timer(void(*subroutine)(),time_t tempo);
time_t timer(void);
float  _ftime(void);
```

CLOCK REALTIME

```
char *get_time(void);
char *get_date(void);
unsigned char get_oro(unsigned char c);
void set_time(char *s);
void set_date(char *s);
void set_oro(unsigned char c,unsigned char val);
```

DERIVATE

```
unsigned char PulseOn(unsigned char *p,unsigned char i);
unsigned char PulseOff(unsigned char *p,unsigned char i);
```

DELAY

```
unsigned char Delay(unsigned long int *h,int n,unsigned char i);
unsigned char DelayUp(unsigned long int *h,int n,unsigned char i);
unsigned char DelayDown(unsigned long int *h,int n,unsigned char i);
```

ONE SHOT

```
unsigned char MonostableUp(unsigned long int *h,int n,unsigned char i);
unsigned char MonostableDown(unsigned long int *h,int n,unsigned char i);
```

Graphical functions

```
void ioinit(int priority);
void crt(int dx,int dy,int displ_cieco);
void display(int dx,int dy,int displ_cieco);
```

```
void crt_res(int dx,int dy);
void v_send(char c1);
void v_stop(void);
void v_cls(void);
void v_locate(int x,int y);
void v_llocate(int y,int x);
void _font(int font_num,int scala_x,int scala_y);
void v_font(int font_num,int scala_x,int scala_y);
void v_setfont(int fontn,int fontx,int fonty);
void v_color(int c1,int c2);
void _color(int bg,int fg);
void v_line(int x0,int y0,int dx,int dy);
void v_linec(int x0,int y0,int dx,int dy,int color);
void v_rect(int x0,int y0,int dx,int dy);
void v_rectc(int x0,int y0,int dx,int dy,int color);
void v_rectfill(int x0,int y0,int dx,int dy);
void v_rectfillc(int x0,int y0,int dx,int dy,int color);
void v_printchar(unsigned char c);
void v_print(char *s);
void v_paint(const unsigned char *s);
void v_paint_ram(unsigned long int ram,const unsigned char *s);
void v_copy(int xs,int ys,int xd,int yd,int dx,int dy);
void v_copy_from_ram(unsigned long int ram,int step,int xd,int yd,int
dx,int dy);
void v_fill_mode(unsigned char mode,int level,int color0,int color1,int
color2);
void lock(void);
void unlock(void);
```

KEYBOARD

```
void v_taston(int colonna,int riga,int nrighe,int ncolonne);
void v_tastoff(void);
void v_spot(unsigned long int leds);
void v_all(unsigned char);
void v_tik(unsigned char x0);
void v_freq(unsigned int KBAUD);
unsigned char v_inkey(void);
void keyon(int lin,int col,int nx,int ny,char *show_keys);
void keyoff(void);
void key_msg(char *msg);
void key_wait(void);
void key_top(void);
void key_release(void);
void keyboard(const char* keyboar_keys);
void key_show(int lin,int col,int nx,int ny,char *show_keys,const char*
keyboar_keys);
unsigned char t_inke(void);
unsigned char t_inkey(void);
void t_input(char * string,short int max);
void t_edit(char * string,short int max);
```

SERIAL PORT

```
int  com_enable(struct COM *com);
int  com_disable(struct COM *com);
int  com_open(struct COM *porta,long int baud);
int  com_close(struct COM *porta);
int  protocol_mode(struct COM *porta,char mode);
int  com_speed(struct COM *porta,long int baud);
int  com_tx_empty(struct COM *porta);
int  com_tx_full(struct COM *porta);
```

```
int  com_tx_size(struct COM *porta);
int  com_rx_empty(struct COM *porta);
int  com_rx_size(struct COM *porta);
int  com_tx(struct COM *porta,unsigned char c);
int  com_rx(struct COM *porta);
int  com_txsl(struct COM *porta,unsigned char *c,int len);
int  com_txs(struct COM *porta,unsigned char *c);
int  bload(struct COM *porta,void *buffer,int len);
int  bsave(struct COM *porta,void *buffer,int len);
int  rxchars(struct COM *porta);
int  com_rxlen(struct COM *porta);
int  com_rxcode(struct COM *porta);
int  onrx(struct COM *porta,void(*subroutine)());
int  ontx(struct COM *porta,void(*subroutine)());
int  polling(struct COM *porta,int numout,int numin,int tempo);
```

FLASH

```
int sf_wdata(unsigned char *src,long int dst,long int num);
int sf_rdata(unsigned char *src,long int dst,long int num);
int sf_cdata(unsigned char *src,long int dst,long int num);
int sf_wcode(unsigned char *src,long int dst,long int num);
int sf_rcode(unsigned char *src,long int dst,long int num);
int sf_ccode(unsigned char *src,long int dst,long int num);
```

PEN USB

```
void cf_init(void);
int  cf_dir(char *filename,char *result,char mode);
int  cf_open(char *filename,char mode);
int  cf_test(void);
int  cf_close(void);
int  cf_del(char *filename);
int  cf_rb(char *buffer,int len);
int  cf_wb(char *buffer,int len);
int  cf_seek(long int posiz);
int  cf_rd(char *buffer);
int  cf_wr(char *buffer);
int  cf_status(int mode);
int  cf_present(void);
```

FILES MAMAGEMENT

```
FILE *fopen(const char *name,const char *mode);
int   fclose(FILE *file);
size_t fread(void *buffer,size_t count,size_t num,FILE *file);
int fgetc(FILE *file);
char *fgets(char *buffer,int max,FILE *file);
int  fscanf(FILE *file,const char *format, ... );
size_t fwrite(const void *buffer,size_t count,size_t num,FILE *file);
int fputc(int c,FILE *file);
int fputs(const char *buffer,FILE *file);
int  fprintf(FILE *file,const char *format, ... );
int  fseek(FILE *file,long offset,int whence);
long int ftell(FILE *file);
int feof(FILE *file);
long int flen(FILE *file);
long int fpointer(FILE *file);
void mount(void);
void unmount(void);
```

CAN

```
void can_time(int t);
void can_perif(int n,int t,int baud);
unsigned char can_busoff(void);
unsigned char can_msg(void);
void can_rxmsg(unsigned char n);
```

CANOPEN

```
void canopen_on(unsigned char sub);
void canopen_off(void);
unsigned char canopen_empty(void);
unsigned char canopen_full(void);
unsigned char canopen_msg(void);
void canopen_flush(void);
unsigned char canopen_rxmsg(void);
unsigned char canopen_txmsg(void);
unsigned char canopen_txmsg_len(unsigned char len);
unsigned char canopen_requestmsg(void);
unsigned char canopen_txsdo(void);
void canopen_onrx(void(*subroutine)());
int can_13_ok(void);
int can_14_ok(void);
void buson(void);
void canopen_standard_sub(void);
unsigned char canopen_perif(unsigned char i,unsigned char type,void
(*subrx1)(),void(*subrx2)());
```

EXPANSIONS I/O

```
void need(int,int,int,int);
void refresh(unsigned int perif,COM *com,long int baud,int numin,int fir-
stin,int numout,int firstout,int timeout);
```

AGAIN

```
void dev_tr80(void);
int  cpufreq(void);
int  CPUFREQ(void);
```

KERNEL

The operating system used in the programs generated from Proteus ES S400 version supports 3 task accessible contenders, to which it is possible to assign of the priorities.
Il task 1 man-machine manages the graphical interface and for default it has priority 4
Il task 2 it manages the cycles of PLC, that one written in Ladder language and that one written in C, which comes alternatively executed. The priority of default is 4.
Il task 3 it is used for the control of the interpolation. (linear-circular), and if this last one is not used, can be used liberations from the programmer. A not advanced priority to 2 is advised.
The operating system has **full preemptive round robin a Time-sharing**, with a settabile Time-slice from 100 usec. to 1 msec. and with a maximum latency time from 0.9 mSec. to 2 mSec.
They are previewed and usable from programmer the classic functions of management of the task (exec, remove, suspend, resume), the semaphore of synchronization and mutual exclusion, and the management of meaningful events (p, v, waitfor, event).
Normally it is not necessary to use some of these functions in a normal Graphical PLC-Interface program, however they exist and are documented, in the event the programmer wanted to use them.

In version ARM (VK3-V8 etc) **32 task** are supported.
In such case the commando exec_task can contain like number of task number ZERO, and the system will automatically assign the number of a task free.
In this version it is possible to execute the commandos suspend_task, remove_task etc passing in the parameter number of task the name of the task same.

Ezample:

```
exec_task(mytask,0,4);
.
.
.
suspend_task(mytask);
```

```
int  exec_task
(void(*fun)(),int numtask,int priority);
```

Execute a task.

Fun is the name of the function without arguments that comes executed like task. It must contain a loop closed and it does not have to never finish, otherwise the result is unforeseeable.
Task the 1 is executed automatically to the start of the machine and the associate function is the main.

If it exists and it is activated the PLC it comes started automatically as task 2 and contains a loop already closed.
If we do not have need of the PLC standard and want to use task 2 defined from we, we must include in COMMON.H the line:

```
#define NO_PLC
```

That it excludes the generation of the code and the inclusion of plc the standard.
Task the 3 are classified to the programmer, except the case in which in the plan they are used of the interpolation objects.

Numtask it is the number of task (the 1, 2 or 3 for S400), (0..32 for ARM) Task the 0 are allotted automatically from the system in a task free.

Priority it is the temporal priority of task or better the number than slot to classified it.
Slot temporal hard 100 uSecondi.
Task 1 and 2 has for default priority 4, and therefore they turn alternatively for 400 uSec everyone. In such a way the time total of timesharing is of 800 uSec. And the maximum latency time, that is the time in which a task it does not turn, is of 400 uSec.

Code of return:

0 = succeeded operation
-1 = error.
It comes generated error and the operation does not come executed if it is tried to execute a task in a slot in which an other is already active task.
So that the operation succeeds is necessary that the slot indicated it is not assigned to some task, or that this last one or suspended or removed.

Example:

In common.h we define task and we disable the PLC

```
#define NO_PLC
// task that ago to blink an out
void lampeg(void)
{
   while(1)
   {
      o1=o1 ^1;
      wait(1 SEC);
   }
}

// task that active an escape 1 sec to the week
void settim(void)
{
   while(1)
   {
      o2=1;
      wait(1 SEC);
      o2=0;
      wait(1 WEEK - 1 SEC);
   }
}
```

From some part in the code of the programmer…

```
// execution of the task lampeg like task 2
priorities 1
// and of the task settim like task 3 priorities 1
exec_task(lampeg,2,1);
exec_task(settim,3,1);
```

```
int  resume_task(int numtask);
```

Enable a stop task.

Numtask it is the number of the slot in which the task is found to start again. (or the name of the task in the version for ARM, provided that the having ruotine such name has not been launch on more task)

The **error code** returns with the value

0 if the operation has gone to good aim
-1 if the slot it is not assigned no task, or is assigned to a task already active or removed.

Att. Suspend and Remove are two similar commandos much, and have both like effect that one of disable a task.
A task suspended with suspend can being definitively removed with remove or enable again with resume.
A task removed with remove it is removed definitively and if it in execution wants itself to be sent back, it is necessary a new one exec and the task it will leave again from the beginning.
A task it can suspend or to remove if same, in the which case, so that the action happens immediately and it does not attend the end of the time of slot, is necessary to make to follow a command of idle () that it forces the reskeduling.
Obviously a task it cannot make resume or the exec of same himself.

Example:

```
// with rif. To the example in exec_task ...
resume(2);      //resume the task lampeg
resume(lamp);   //only ARM
```

```
int  suspend_task(int numtask);
```

Suspend a task.

Numtask it is the number of the slot in which the task is found to suspend. (or the task name in the version for ARM, provided that the having ruotine such name has not been launch on more task)

The **error code** returns with the value
0 if the operation has gone to good aim
-1 if the slot it is not assigned no task, or is assigned to a task already suspended or removed, or if the task it engages of the semaphor of resources. (as an example, nobody can suspend or to remove a task that it has classified the video or pen USB, otherwise such resources would remain blocked in order always)

att. Suspend and Remove are two similar commandos much, and have both like effect that one to disable a task.
A task suspended with suspend it can be definitively removed with remove or riattivato with resume.
A task removed with remove it is removed definitively and if we wish restart, it is necessary a new one exec and the task it will leave again from the beginning.
A task it can suspend or to remove if same, in the which case, so that the action happens immediately and it does not attend the end of the time of slot, is necessary to make to follow to aforesaid a command of **idle ()** that it forces the reskeduling.
Obviously a task it cannot make resume or the exec of same himself.
Example:

```
// con rif. all' example in exec_task ...
suspend(2);  //suspend the task lampeg
```

```
int   remove_task(int numtask);
```

Remove a task.

Numtask is the number of the slot in which the task is found to remove (or the name of the task in the version for ARM, provided that the having ruotine such name has not been launch on more task)

The **error code** returns with the value:
-0 if the operation has gone to good aim
-1 if the slot it is not assigned no task, or is assigned to a task already removed, or if the task it engages of the semaphors of resources (for because you see suspend_task).

Att. Suspend and Remove are two similar commandos much, and have both like effect that one to disable a task.
A task suspended with suspend it can be definitively removed with remove or enable again with resume.
A task removed with remove it is removed definitively and if it in execution wants itself to be sent back, it is necessary a new one exec and the task it will leave again from the beginning.
A task it can suspend or to remove if same, in the which case, so that the action happens immediately and it does not attend the end of the time of slot, is necessary to make to follow to aforesaid a commando of idle () that it forces the reskeduling.
Obviously a task it cannot make resume or the exec of same himself.

When a task it is removed from a slot, to all the effects are as if in the slot it had not never turned some task.

Example:

```
// with ref. To the example in exec_task ...
remove(2);  //remove the task lampeg
```

```
void _idle(void);
```

Go to the next task

This command makes to pass the Time-sharing immediately to the task successive, without to attend the end of the slot temporal to assigned he.
He can be used to have more speed on the machine, in the event in which we are attending that he succeeds an event and we do not want to engage the time uselessly machine in the wait.

Example:

```
// attended release
while(!i22)
{
   _idle();
}
// continuation of the program
```

We attend that income 22 active in order to continue itself. In the wait the time for to the others task.

```
int  _p(int event);
int  _v(int event);
```

They are the semaphors of mutual exclusion. They always travel in brace.
Event is the number of a resource where we wish to make access.

p (event) it reserves the resource, if this is occupied stop own task, otherwise continues.
v (event) free a resource previously reserved and ago the resume of eventual others task that they have been suspended waiting for such resource.

They return with **code of error 0** if the operation has had happened.
p (event) it returns with **code of error - 1** if the semaphor already it is reserved from own task.
v (event) it returns with code of error - 1 if the semaphor it is free or if it is reserved from an other task task. Only task that it has reserved a resource has the right to free it.
The operating system of Proteus 2007 can manage **32 semaphors**.
Semaphor the 1 is classified for the CRT (sees lock and unlock)
Semaphor the 2 are classified to pen USB (see mount, unmount)
The semaphors from 3 to 31 are usable liberations from the programmer.

Example:

```
// in task 1...
_p(5);
v_font(OFF8,1,1);
v_color(1,254);
v_llocate(3,3);
v_print("hello !");
_v(5);

// in task 2...
_p(5);
v_font(OFF16,1,1);
v_color(0,100);
v_llocate(3,23);
v_print("ciao !");
_v(5);
```

```
void _waitfor(int event);
void _event(int evento);
```

Synchronization between task

Event is the number of an event (a number comprised between 1 and 127).

These two functions concur the synchronization between they of two task various.
_waitfor it suspends the task putting it waiting for the last event like argument.
_event it ago declares the last event like argument and the resume of the eventual ones task that they are suspended waiting for such event. If they were some, immediately passes the control to the first one of they, without to attend the end of temporal own slot.
Cases particular:
_waitfor (0) disable the interrupt of system
_waitfor (254) suspend itself the task.
-waitfor (255) remuve itself the task.
_event (the 254) ago resume of all task suspended

Example:

```
// in task 1…
While(timo)
{
  _waitfor(33);
  v_font(OFF8,1,1);
  v_color(1,254);
  v_llocate(3,3);
  printf("timeout %d",timo);
}

// in task 2...
timo=xx;
_event(33);
```

Task the 1 in is attended that timo goes to zero and sospende itself. Task the 2, when it changes the variable timo, notification the change through event 33 starting task the 1.

ENCODERS

*The **PLC** of the **S400** series have till 4 controls axis to encoders inner, expandible to 16 (**) with additional modules.*
They support encoders increases them bidirectional with zero ref. operating till frequencies of 1 Mhz. (10KHz. In zero set).
The axis from 1 to 4 are enable with the declaration load_asseN_fast
The axis besides the quarter are managed in transparent way to the software, and come enabled with the declaration can_perif (see understood it CAN).

The quotas, are double word with sign, and the system defines following the variable ones automatically:

encoder1...encoder16	the quote (pulse encoder)
enczero1...enczero16	the quote of zero (*)
searching_z[0...15]	flag find zero actived (read only)
search_z[0...15]	flag start find zero (write only)

To the inner axis we can also associate a name of the various quota (if we do not use the routines automatic of zero setting on zero ref.) and eventually we can also write of the routines personalized for the interrupts.
The single thing that diversifies the inner axis from those exteriors is the various way in order to declare them. Once declared, they come dealt exactly in the same way.

(*) draft of a variable one of support, than does not influence the read value of the quota, but it is only a memorandum, usable if we want like offset in order calculating the real value of the quota.

(**) Expandible to 32 with modules can-open.

```
void load_asse1_fast(volatile long int
*quota);
void load_asse2_fast(volatile long int
*quota);
void load_asse3_fast(volatile long int
*quota);
void load_asse4_fast(volatile long int
*quota);
```

Enable inner axis

Quote variable **long int** associated to the quote

With these declarations, than of habit they are found in rows **START.C**, they come defined and activated the axis to encoder inner to the main module S400 (what it contains the program).
If we want to use routines of zero setting the axis on zero ref, for the variable quote we must use the variable ones of system **encoder1, encoder2, encoder3 and encoder4** respective.
Wanting, we can declare and use for the quotas any variable one, provided that of type **long int**.

If in the plan a greater number of zero aces is declared, the initialization of the declared axis is made automatically from the compiler and it is not necessary that the programmer it gives the command load_asseX_fast.

Example:

```
// in start.c
load_asse1_fast(encoder1);
load_asse2_fast(encoder2);
load_asse3_fast(encoder3);
load_asse4_fast(encoder4);
```

```
void load_asse1(void(*)(),void(*)());
void load_asse2(void(*)(),void(*)());
void load_asse3(void(*)(),void(*)());
void load_asse4(void(*)(),void(*)());
```

Routines of interrupt personalized

With these commandos who of habit find themselves in file **START.C**, we can define of the routines of interrupt personalized for the encoders. To every impulse UP of the encoder the function declared in the first parameter will be called.
To every impulse DOWN of the encoder the function declared will be called in to second parameter.
The functions must be without parameters and variable of return (void (*) ()).

ATTENTION: This management of the encoders can remarkablly reduce the maximum frequency of job of the same ones.
THIS FUNCTION IS NOT USABLE IF THE PROGRAM MUST TURN ON PC.

Example:

```
// in COMMON.H…
void up1(void)
{
  if (encoder1<100) encoder1++;
}
void dn1(void)
{
  if (encoder1>0) encoder1—;
}

// in START.C
load_asse1(up1,dn1);
```

In this example we manage the quota limited axis 1 between 0 and 100. To what it serves? To nothing, it is an example.

```
void find_zero(unsigned char n,long int q);
```

It find zero ref.

n number of axis (1..16)
q value to assign to enczero of the axis.

This command set the flag search_z of the corresponding axis. The flag **searching_z** of corresponding axis (**less 1, you see example**) goes to 1 and he remains till when the zero ref has been found. When this happens the quote the axis (encoderX) goes to **ZERO**.
The flag searching_z, if to 1 it indicates that the search is in course, but in order to make a control we must attend reskeduling for the inner axis and at least the 5 mSec. For those exteriors, otherwise we cannot know, if such flag it is worth zero, if the search already is ended or not still begun. It is not said moreover that such flag it goes to 1, as if the command comes given when the encoder he is already on the zero ref, we will not never see it to 1.

The correct sequence is therefore:
1– find_zero (.)
2- idle () (obligatory if there is 3)
3 - the while one (searching_z [.]) {} (optional)

So that the ref of zero or found, is necessary that hard at least 100 uSec., that is that the 10khz frequency of the encoder does not exceed .

Example:

```
// find zero ref axis 1
find_zero(1,1000);
idle();
while(searching_z[0]);
// find zero ref axis 12
find_zero(12,0);
wait(5);
while(searching_z[11]);
```

Value 1000 of q goes in enczero1.
When the zero ref is found encoder1 goes to ZERO.

```
void clear_asse(unsigned char n);
```

Clear the quote of a axis

n number of the axis (1..16)

This function is usable for inner and external axis.
In order to clear the quote an axis the variable one is not sufficient to put to zero relative encoderX, as this is controlled from the hardware.
Only for the inner axis, and only if the program does not turn on PC, it is possible to assign a value to a quote in the following way:

```
IEN=0;
encoderX=valore;
IEN=1;
```

In all the other cases, we can only put to zero the quote by means of the instruction clear_asse.
We must see the quotas encoderX like pulse counters of a encoder absolute, rather than like real quotas, and use offset and a multiplication factor for having the quote in physical units.

Example:

```
// clear axis 1
azzera_asse(1);
// clear axis 12
azzera_asse(12);
```

INTERRUPTS (only for S400)

The PLC S400 manages a sure number of resources by means of routines of interrupt. The programmer it can personalize such routines replacing own code to that one of default. Draft of an operation much delicate one and for expert programmers, can it is unknown be given place to malfunctioning of the program and to the block of the machine if exactly what it is being made.
The detailed explanation of the structure of the interrupt of the cpu of the S400 is outside from the attempts of this manual, for which it is sent back to the user manual of cpu (c167cs_um_v2.0_2000_07.pdf) the liberations releasable from the site
www.keil.com/dd/docs/datashts/infineon/**c167cs_um**.pdf

ATTENTION: Using routines personalized to interrupt us alloy to the detail hardware of the PLC, what that is outside from the philosophy of Proteus, create programs that cannot turn on PC and that sure they will not be compatible with new release of the PLC S400.

`void load_interrupt(int num,void(*)());`

Charge the carrier of interrupt of the ruotine in the operating system.

example

```
#define T0TRAP 0x20
static interrupt (T0TRAP) using (_RBANKT0) void
t0_int( void )
{
    step_passo=step_passo ^ 1;
    o18=step_passo;
    if (!step_passo) step_quota--;
   _OUT2=_lout2;
    if (!step_quota) T0IE=0;
}
load_interrupt(T0INT,t0_int);
T01CON=(T01CON & 0xff00) | 0x40;
T0IC=0x2c;
```

ANALOG IN OUT

The inputs and outputs are dealt from the program like variable that come managed from the operating system, which, at a convenient time, send the present values physically them to the I/O.
The delay, from the change of a variable one to the change of the state of the in-out is inferior to the millisecond (maximum 2 mSecondi for the I/O of the expansions) and with appropriate instructions we can force the immediate refresh.

Analog in: **pot1,pot2**… or **Pot[0],Pot[1]…**
Analog out: **anal1,anal2**.. or **Anal[0],Anal[1**]...

The two forms are equivalents.
The S400 controller has 4 inner analogic outputs +12 external with the modules of expansion, operating in range - 32000… +32000.
+24 exteriors with the modules of expansion have moreover 8 analogic input insides, operating in range 0… 4095.

```
void set_anal(int num,int level);
```

It physically changes the level of an analogic out

num number of the analogic out
level value of the analog out

This command physically changes the level of the analogic out (inner) and refresh the associated value of the variable one.
He uses himself in the event in which a or important immediate refresh of the level.

Example:

```
void refresh_anal(int num)
{
   int i;
   num++;
   for (i=1;i<num;i++)
     if(anol[i]!=anal[i]) set_anal(i,anal[i]);
}
```

The example is the insert in library of the command refresh_anal (sees page following), which uses set_anal in order to refresh the analogic outputs that are changed.

```
void refresh_anal(int num);
```

It physically executes the refresh of the analogic outputs that are changed.

num number of inner analogic outputs (= 4)

This ruotine is called automatically from operating system approximately 1 turns every millisecond, for which, if not there are reasons for a refresh more express, it is not necessary to call it in the program.
With the exception of the analogic exited ruotine set_anal this last one can refresh more, and only those which are changed.
In the page precedence we can see the list one of this function.

Example:

```
// ----------------
anal1=i1*1000-500;
anal2=i2*1000-500;
anal4=i3*1000-500;
refresh_anal(4);
// ----------------
```

In this case refresh_anal it only executes the physical refresh when the input types them relative changes of level.

```
void pot_init(void);
```

Set interrupt and the DMA for the reading of potenziometers (the analogic in)

This ruotine is used inner from the operating system and it does not have to never be used in the program.

```
void pot_start(void);
void pot_stop(void);
```

Start and stop of the reading of the potenziometers.

The reading of the potenziometers is a slow operation from the point of view of the time machine, for which it is previewed of being able to make it to leave when it serves and of stop then when not servants more.

On the start of the machine, the reading is disabled, for which, for being able to read the potenziometers, a command of pot_start is necessary. Of usual this commando finds itself in START.C.

Example:

```
// ----------------
pot_start();
// ----------------
```

```
void pot_refresh(void);
```

It refreshes the reading of the potenziometers.

This command executes the average on the values of reading of the potenziometers and refresh the value of the variable ones of the analogic inputs (pot [n]).
He comes called periodically from the operating system, and he does not have to never be used in the program.

DIGITAL IN OUT

The inputs and outputs are dealt from the program like variable that come managed from the operating system, which, at a convenient time, send the present values physically them to the I/O.
The delay, from the change of a variable one to the change of the state of the in-out is inferior to the millisecond (maximum 2 mSecondi for the I/O of the expansions) and with appropriate instructions we can force the immediate refresh.

Digital inputs: **i1,i2...**
Digital outputs: **o1,o2...**

The S400 controller has 32 outputs inner + 96 external with the expansion modules.
It have moreover 32 analogic inputs insides +96 exteriors with the expansion modules .

The inputs and outputs come seen as variable predefined of type bit i1… i128 and o1… o128.

With ulterior modules RS-232 and can-open we can arrive to a maximum of 160 inputs and 160 outputs directly (i160, o160) and till 2048 inputs and 2048 outputs managed to groups of 8.

Example:

```
// ————————————————
if ( i1 & !i2) o1=1;
o2=i3 | i4;
// ————————————————
```

```
void __i(void);
void __o(void);
```

Immediate Refresh of the I/O

In case for scopes details we wanted an immediate modernization of the outputs, we must give the command __o () after the expressions that change the outputs.
In the same way if we want to read immediately the state of the inputs, we must give the commando __i () before examining the inputs.

Such commandos act alone on the 32 incomes and escapes to edge of the S400 and **are not usable for programs that turn on the PC.**

Example:

```
// ------------------
__i();
if ( i1 & !i2) o1=1;
o2=i3 | i4;
__o();
// ------------------
```

```
unsigned char I(unsigned int n);
unsigned char O(unsigned int n);
void Out(unsigned int n,unsigned char x);
```

These commands concur the reading and the refresh of the i/o types them in addressable form

Example,

```
if ( i1 & !i2) o1=1;
o2=i3 | o4;
```

It can be also written:

```
int x1=1,x2,2,x3=3,x4=4;
if ( I(x1) & !I(x2) ) out( x1 , 1 );
Out( x2 , I(x3) | O(x4) );
```

TIMERS

The operating system of Proteus 2007 manages a system clock with resolution of 1 mSec. with which times of waited for or executing of the functions to regular cadence can be created.

The minimal interval of gestibile time is 1 millisecond.
The maximum interval of gestibile time is of 2^31-1 milliseconds to equal 24 days + 20 hours + 31 minuteren + 23 second ones, that it is the time of the maximum cadence of repetition of a ruotine to time or the maximum time of the wait.

The time, where not various specified, comes passed in mSec. (10 = 10 mSec.) and the following words can be used key in order to specify unit of various measure of time:

MSEC **millisecond/s**
SEC **second/s**
MIN **minute/s**
HOUR **hour**
DAY **day**
WEEK **week**
HOURS **hours**
DAYS **days**
WEEKS **weeks**
ONCE **once**
NOW **now**

As an example they are correct expressions:

```
23 SEC
1 DAY + 3 HOURS + 18 MSEC
2 WEEKS + 1 HOUR + 5 SEC
```

The operating system for the S400 series is in a position to managing at the same time till an active maximum of 32 routines to time, while for series ARM they can be managed at the same time till 100 routines to time active.

```
int exec_timer(
void(*subroutine)(),
time_t interval,
time_t fra_quanto);
```

It executes a ruotine to time specifying the cadence and the moment first execution.

Subroutine it is the name of a function without arguments and code of return, that it must be executed periodically. A reasonable time must last, in order not to block the system.

Interval is the time of repetition. If it must be executed a single time, use for this parameter the value OUNCES or 0.

Fra_quanto is the execution delay. If it wants the immediate execution (that is to expiring of the running millisecond) to use value NOW or 0.

This command can be used in order to send in execution a ruotine to time, or in order to change the time of execution or in order to already delay a ruotine to time in execution.

Example:

```
// routines to time
void blink(void) {o1=!o1;}
void enable(void) {o2=1;}
// . . . .
// from some part in the program…
// executes blink to active cadence 1 according
// enable o2 between 1 hour
exec_timer(blink,1 SEC, 10 MSEC);
exec_timer(enable,ONCE, 1 HOUR);
// . . . .
// it changes the cadence to 1/2 sec.
exec_timer(blink,500 MSEC, NOW);
```

```
int  resume_timer(void(*subroutine)());
```

It restores a ruotine to time

Subroutine: name of the ruotine to time, previously sent in execution with the command exec_timer

This commando restores the normal execution of the specified ruotine to time, with the cadence previously declared in the command exec_timer.

Example:

```
// routine to time
void blink(void) {o1=!o1;}
// . . . .
// from some part in the program…
 // executes blink to cadence 1 second
exec_timer(blink,1 SEC, 10 MSEC);
// . . . .
// it changes the cadence to 1/2 sec.
exec_timer(blink,500 MSEC, NOW);
// . . . .
// suspend blink
suspend_timer(blink);
// . . . .
// resume blink
resume_timer(blink);
// . . . .
// suspend blink for a 1 minute
sleep_timer(blink, 1 MIN);
// . . . .
// remove blink
remove_timer(blink);
```

```
int  suspend_timer(void(*subroutine)());
```

It suspends a ruotine to time

Subroutine: name of the ruotine to time, previously sent in execution with the command exec_timer

This command suspends to indefinite time the normal execution of the specific ruotine to time. The ruotine remains however active and continues at the same time to occupy its place in the declarable carrier of the 32 routines to time.
If we think that it does not have more being sent in execution, it is best to give a command of remove_timer in order making place to eventual others routines time.

Example:

```
// routine to time
void blink(void) {o1=!o1;}
// . . . .
// from some part in the program…
 // executes blink to cadence 1 second
exec_timer(blink,1 SEC, 10 MSEC);
// . . . .
// it changes the cadence to 1/2 sec.
exec_timer(blink,500 MSEC, NOW);
// . . . .
// suspend blink
suspend_timer(blink);
// . . . .
// resume blink
resume_timer(lampeggia);
// . . . .
// suspend blink for a 1 minute
sleep_timer(blink, 1 MIN);
// . . . .
// remove blink
remove_timer(blink);
```

```
int   remove_timer(void(*subroutine)());
```

It removes a ruotine to time

Subroutine: name of the ruotine to time, previously sent in execution with the commando exec_timer

This command removes a ruotine to time from the list of the 32 routines to time that can coexist at the same time.
The ruotine has acquitted its task, free place to the others.

Example:

```
// routine to time
void blink(void) {o1=!o1;}
// . . . .
// from some part in the program…
 // executes blink to cadence 1 second
exec_timer(blink,1 SEC, 10 MSEC);
// . . . .
// it changes the cadence to 1/2 sec.
exec_timer(blink,500 MSEC, NOW);
// . . . .
// suspend blink
suspend_timer(blink);
// . . . .
// resume blink
resume_timer(lampeggia);
// . . . .
// suspend blink for a 1 minute
sleep_timer(blink, 1 MIN);
// . . . .
// remove blink
remove_timer(blink);
```

```
int  sleep_timer(void(*subroutine)
(),tempo);
```

Subroutine: name of the ruotine to time, previously sent in execution with the command exec_timer

This command suspends for a sure time a ruotine to time.
Repeating many times over such command the times of suspension THEY DO NOT ADD THEMSELVES but HE IS WORTH the LAST one. Evidently, if we repeat this command with more frequent cadence of the set up time of suspension, the ruotine will not never come executed. We can use this method in order to create a similar WATCHDOG, SAVESCREEN or functions.

Example:

```
// routine to time
void blink(void) {o1=!o1;}
// . . . .
// from some part in the program…
 // executes blink to cadence 1 second
exec_timer(blink,1 SEC, 10 MSEC);
// . . . .
// it changes the cadence to 1/2 sec.
exec_timer(blink,500 MSEC, NOW);
// . . . .
// suspend blink
suspend_timer(blink);
// . . . .
// resume blink
resume_timer(lampeggia);
// . . . .
// suspend blink for a 1 minute
sleep_timer(blink, 1 MIN);
// . . . .
// remove blink
remove_timer(blink);
```

```
int  wait(time_t n);
```

It suspends for a sure time task the current.

n time of suspension.

This command suspends for a sure time the task in which he comes executed. For the unit times you at the beginning see of the chapter it TIMERS.

Example:

```
// o1 enable for 100 mSec.
o1=1;
wait(100 MSEC);
o1=0;
```

```
int   every(time_t n);
```

It suspends for a sure time task the current in order to fix the duration of a loop.

n cadence.

This command suspends for a sure time the task in which he comes executed if precedence is not passed to the time specified from the time at least.
For the unit times you at the beginning see of the chapter TIMERS.

Example:

```
// loop1.
for (i=1;i<100;i++)
{
  wait(90 MSEC);
  o1=1;
  wait(10 MSEC);
  o1=0;
  … altre cose
}
// loop2.
for (i=1;i<100;i++)
{
  every(100 MSEC);
  o1=1;
  wait(10 MSEC);
  o1=0;
  … altre cose
}
```

With the exception of loop the 1, loop the 2 exactly have a time of cycle **100 milliseconds**, as every () attend the time lacking before continuing.

```
time_t _time(time_t *dst);
float  _ftime(void);
```

It reads the system timer.

The argument dst is optional. If not used to put 0.

This commando reads the variable one unsigned long int system timer.
The first version (_time) is valid for PC and S400.
The version floating point, valid for S400, gives to the value of the time with greater resolution (microsecond) but it reduces the maximum time from 2^31-1 to 2^24-1 milliseconds.
It can serve in order to measure the time of execution of a ruotine, or for other scopes.

Example:

```
// how much is the time of execution of SIN?
volatile float arg=1.2345,ris;
int i;
t1=_time(0);
for (i=0;i<5000;i++)
   ris=sin(arg);
t1=_time(0)-t1;
printf("sin(arg) dura %f mSec",(float)t1/5000.);
```

In the example we have used the prefixed one flown them for ris in order forcing the execution of the function, excluding the optimization.
A loop has been used of 5000 for having a sufficient precision.
.

CLOCK REALTIME

The operating system of Proteus 2007 manages a usable clock-calendar for statistics, control of productivity, or simply in order to show on the screen that now is.

The program in C communicates with the clock through the commandos who we will see in the successive pages, exchanging the data by means of strings of eight characters, in the format:

HH.MM:SS for the hours
e
GG-MM-AA for the days (serie S400)
GG-MM-AAAA for the days (serie ARM)

Moreover two functions are previewed (get_oro and set_oro) that they can read and write the single byte of the clock. These two functions are valid only for programs on S400 and they do not guarantee compatibility'con the successive versions of the compiler.

```
char *get_time(void);
```

It reads the hour

This function returns with the address to a string of eight characters ascii that it contains the hour puts into effect them in the moment in which has been executed the function, with format **HH.MM: SS.**

Example:

```
// it up prints the hour on the left on the screen
v_font(OFF8,1,1);
v_color(0,255);
v_llocate(1,1);
printf("%s",get_time());
```

```
char *get_date(void);
```

It reads the today's date

This function returns with the address to a string of eight characters ascii (ten characters in version ARM) that it contains the date puts into effect them in the moment in which has been executed the function, with format **GG-MM-AA.**

Example:

```
// up prints the date and the hour
// on the left on the screen
v_font(OFF8,1,1);
v_color(0,255);
v_llocate(1,1);
printf("%s  %s",get_date(),get_time());
```

```
void set_time(char *s);
```

set the hour

s string with the hour in format **HH.MM: SS**

This function refresh the clock with the values contained in the last string like parameter.
It is possible to refresh also only to the hour or and minuteren, now passing a shorter string.
For the characters of separation between hours, second minuteren and we can use any character ascii.

Example:

```
// set hour and minutes
// to noon and thirty
set_time("12-30");
```

```
void set_date(char *s);
```

set the date

s string with the hour in format **GG.MM: AA (GG-MM-AAAA in version ARM)**

This function refresh the clock with the values contained in the last string like parameter.
It is possible to refresh also only to the day or day and month, passing a shorter string.
For the characters of separation between day, month and year we can use any character ascii.

Example:

```
// set the date to 24 May
// to noon and thirty
set_time("12-30");
set_date("24-05");
```

```
unsigned char get_oro(unsigned char c);
void set_oro(unsigned char c,unsigned
char val);
```

They are two functions that can read and write the single byte of the clock. These two functions are valid only for programs on S400 and they do not guarantee the compatibility with successive versions of the compiler.

C position of the character in the clock of S400
V value to write.

The single ones bytes contain codified exadecimals values in BCD

C assigned
0 seconds
1 minutes
2 hours
3 day
4 mount
5 —
6 yar

Example:

```
// it read and set the seconds unsigned char sec;
sec=get_oro(0);     // it read the seconds in BCD
sec-=(sec>>4)*6;    // it cenge in number
set_oro(0,0x35);    // set the seconds to 35
```

DERIVATIVE

```
unsigned char PulseOn(unsigned char
*p,unsigned char i);
unsigned char PulseOff(unsigned char
*p,unsigned char i);
```

Derivative

p variable unsigned char of service
i variable unsigned char of input

These functions give like result **value 1** if the variable one of income is changed regarding the time precedence in which the function it has been called, otherwise gives **value 0**.
PulseOn reveals changes from **0 to 1** of the variable one, while **PulseOff** reveals changes from **1 to 0**.

The variable one p, that it must be declared from the programmer and opportunely inizializzata to a value begins them, servants to memorize the state precedence and must be variable a various one for various inputs. As input agrees a any variable of type byte or bit, and not necessarily a physical input of the PLC.

Example:

```
// counter edge positive in 2
int contai2(void)
{
   static unsigned char h2=0;
   static int conta=0;
   conta+=PulseOn(&h2,i2); // count edge pos. I2
   return conta;
}
```

If call enough frequently reveals the edge of raising of i2.

DELAYED

```
unsigned char Delay(unsigned long int
*h,int n,unsigned char i);
unsigned char DelayUp(unsigned long int
*h,int n,unsigned char i);
unsigned char DelayDown(unsigned long int
*h,int n,unsigned char i);
```

Delayeds of signal

h variable of service unsigned long int
n time of delay in milliseconds
i variable to delayed

These functions delay of a constant time the value of a variable one.
Delay is a real delayed .
DelayUp it delays the transition from 0 to 1
DelayDown it delays the transition from 1 to 0

Of norm they are used in the task of PLC, but they can be used also elsewhere, provided that they come calls regularly and enough frequently. **They cannot be used** if the PLC cycle is not active (if it has been declared #define **NO_PLC**).
The variable one p, that it must be declared from the programmer, servants to memorize the input state and must be variable a various one for various inputs.

Esempio:

```
// counta input 2 (only pulses > 230 mSec)
int contai2(void) {
   static unsigned char h2=0;
   static unsigned long int tt2=0;
   static int conta=0;
   char tmp;
   tmp=DelayUp(&tt2,230,i2); // delay i2 up
   conta+=PulseOn(&h2,tmp ); // count edges pos. I2
   return conta; }
```

MONOSTABILI

```
unsigned char MonostableUp(unsigned long
int *h,int n,unsigned char i);
unsigned char MonostableDown(unsigned
long int *h,int n,unsigned char i);
```

One-shots

h variable of service unsigned long int
n time of delay in milliseconds
i variable to delay

One-shotUp it gives value 1 when the variable one goes to 1, gives value 0 after the indicated time that the variable one is returned to 0.
gives value 0 when the variable one goes to 0, gives value 1 after the indicated time that the variable one is returned to 1.

Of norm they are used in the task of PLC, but they can be used also elsewhere, provided that they come calls regularly and enough frequently. **They cannot be used** if the PLC cycle is not active (if it has been declared **#define NO_PLC**).
The variable one p, that it must be declared from the programmer, servants memorizzare the input state and must be variable a various one for various inputs.

*E*xample*:*

```
// o2=1 for 1 second on gone up on rise-edge of i2
 // i2 on condition that it is present at least 200 mSec.
void repet_i2(void) {
    char temp;
    static unsigned char dd2=0;
    static unsigned long int tt1=0;
    static unsigned long int tt2=0;
    tmp=DelayUp(&tt1,200,i2);          // delay
    temp=PulseOn(&dd2,temp);           // derivative
    o2=One-shotUp(&tt2,1 SEC,temp);    // one-shot
}
```

GRAPHICAL FUNCTIONS

The functions described here allow to define the resolution of the monitor, to set he, to write and to design directly on the Canvas of the screen.
The screen is seen like a matrix XY of points.
The point of coordinates 0,0 is that one up on the left.

Case S400 e PC:
The width of the canvas of the screen is of 1024 pixels. The height depends from the memory amount istallata video. Generally:
1024 pixels for monitot 320x240
2048 pixels for monitot 640x480
3072 pixels for monitot 800x600
5120 pixels for monitot 1024x768
The visible area is always that up on the left. The remaining portion is used for copy operations, for the animations, like buffer for the bitmap and comes managed from the operating system.

Case ARM:
The width of the canvas of the screen is equal to the horizontal resolution oft he display.
The height of the canvas of the screen is equal to the vertical resolution of the display.
They come managed two canvas, and at any moment we can decide which canvas to visualize and on which writing.
This can turn out particularly useful for animations without flickering (technical of the double buffer. For ulterior explanations you see example DX in the section EXAMPLES ARM)

If more task they approach the video, is necessary that the activities are managed from a semaphore, to avoid that the several commands stir themselves, giving place to visualization errors. To such scope the commands of LOCK and UNLOCK can themselves be used.
Except rare cases, like as an example if we must trace of the diagrams or construct a new component, such functions are managed inner from the operating system, and the programmer, using Proteus with the programming to objects, it is not forced to use the graphical functions directly.

```
void ioinit(int priority);
void crt(int dx,int dy,int displ_cieco);
void display(int dx,int dy,int
displ_cieco);
void crt_res(int dx,int dy);
void v_send(char c1);
void v_stop(void);
void v_paint_ram(unsigned long int
ram,const unsigned char *s);
void v_copy_from_ram(unsigned long int
ram,int step,int xd,int yd,int dx,int
dy);
void v_paint(const unsigned char *s);
void v_copy(int xs,int ys,int xd,int
yd,int dx,int dy);
void v_fill_mode(unsigned char mode,int
level,int color0,int color1,int color2);
```

They are inner, valid only for version V5/V10 of the compiler, not usable functions directly from the programmer.

```
void v_cls(void);
```

It clears the visible area of the screen.

This commando colors the screen with the color of the bottom (sees v_color)

Example:

```
// it clears the screen coloring it of blue
v_color(BLUE,YELLOW);
v_cls();
```

```
void v_locate(int x,int y);
```

Set the cursor.

This command set the cursor to established coordinates.

Example:

```
// ---------------------
// in start.c
// ---------------------
v_color(BLUE,YELLOW);
v_cls();
v_font(OFF8,1,1);
v_color(TRASP,RED);
v_locate(12,12);
v_print("prova");
v_color(TRASP,YELLOW);
v_locate(13,13);
v_print("prova");
while(1);
```

```
void v_llocate(int riga,int colonna);
```

Set the cursor.

This command set the cursor to established coordinates. With the exception of v_locate, the coordinates are inverts, and are expressed in characters (multiple of 8). To es. v_llocate (2,5) it is equivalent to v_locate (40,16).

Example:

```
// ---------------------
// in start.c
// ---------------------
v_color(BLUE,YELLOW);
v_cls();
v_font(OFF8,1,1);
v_color(TRASP,RED);
v_llocate(2,2);
v_print("prova");
v_color(TRASP,YELLOW);
v_llocate(3,2);
v_print("prova");
while(1);
```

```
void _font(int font_num,int
scala_x,int scala_y);
```

It selects font of writing and the scale.

This commando selects font of writing and the format of zoom x and y.
Fonts the usable standards directly are:

off8 (OFF8)	0	abcdABCD01234
off16 (OFF16)	1	abcdABCD01234
off24 (OFF24)	2	abcdABCD01234
tim8 (TIM8)	3	abcdABCD01234
tim16 (TIM16)	4	abcdABCD01234
tim24 (TIM24)	5	abcdABCD01234
tim32 (TIM32)	6	abcdABCD01234
MS_Sans_Serif_8	7	abcdABCD01234
MS_Sans_Serif_10	8	abcdABCD01234
MS_Sans_Serif_12	9	abcdABCD01234
MS_Sans_Serif_14	10	abcdABCD01234
MS_Sans_Serif_18	11	abcdABCD01234
MS_Sans_Serif_24	12	abcdABCD01234
Arial_Black_8	13	**abcdABCD01234**
Arial_Black_10	14	**abcdABCD01234**
Arial_Black_12	15	**abcdABCD01234**
Arial_Black_14	16	**abcdABCD01234**
Arial_Black_18	17	**abcdABCD01234**
Arial_Black_24	18	**abcdABCD01234**
Courier_New_8_bold	19	abcdABCD01234
Courier_New_10_bold	20	abcdABCD01234
Courier_New_12_bold	21	abcdABCD01234
Courier_New_14_bold	22	abcdABCD01234
Courier_New_18_bold	23	abcdABCD01234
Courier_New_24_bold	24	abcdABCD01234

The fonts offxx and timxx they are with zoom, the others not, for which for the fonts successive to 6 last the two parameters do not have infuence.

Example:

```
// ---------------------
// in start.c
// ---------------------
v_color(BLUE,YELLOW);
v_cls();
_font(off8,1,1);
v_color(TRASP,RED);
v_llocate(2,2);
v_print("prova");
v_color(TRASP,YELLOW);
v_llocate(3,2);
v_print("prova");
while(1);
```

```
void v_font(int font_num,int scala_x,int
scala_y);
void v_setfont(int fontn,int fontx,int
fonty);
```

As font (you see previous page)

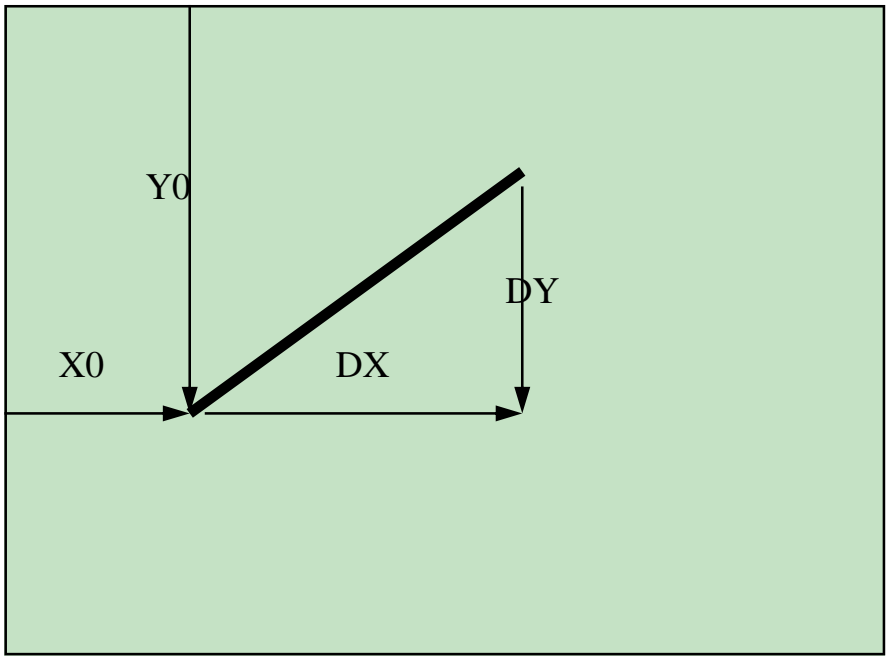
These commandos exist for historical reasons for compatibility with the versions precedence. It is better to use _font (.)

```
void v_color(int bg,int fg);
void _color(int fg,int bg);
```

It sets up the color of the bottom and the character.

bg background color
fg charatter color

This command set the colors to use in order to write on video. The colors are numbers from 0 to 255, with palette fix to 256 colors RGB 3-3 -2.
They exist of the mnemonic ones for the fundamental colors:

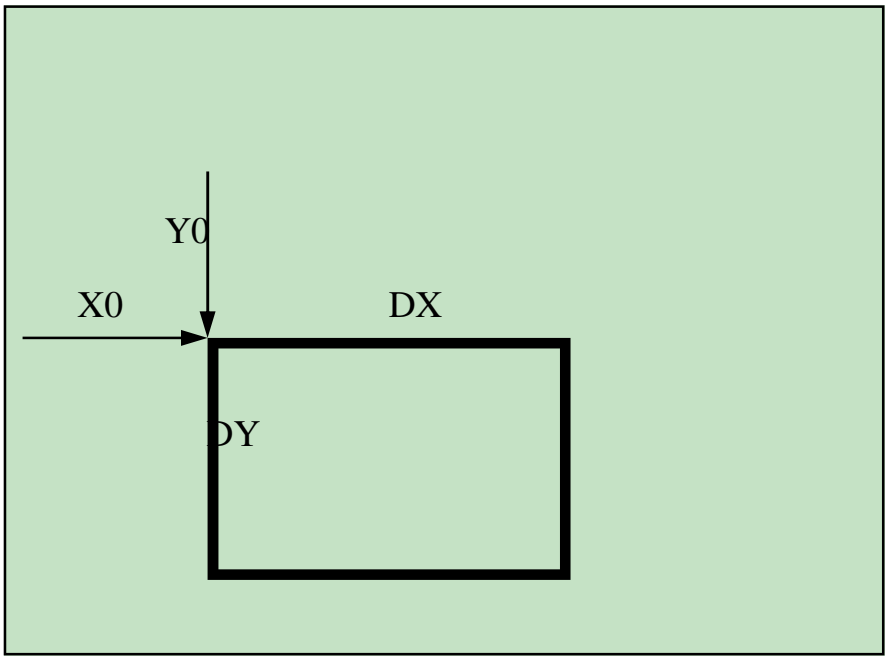


BLACK 0
DARKBLUE 2
DARKRED 128
DARKMAGENTA 130
DARKGREEN 16
DARKCYAN 18
DARKYELLOW 144
DARKGRAY 73
GRAY 146

BLUE	3
RED	224
MAGENTA	227
GREEN	28
CYAN	31
YELLOW	252
WHITE	255
TRASP	**4 (trasparent)**
COMPL	**32 (complement the color)**

Example:

```
// ---------------------
// in start.c
// ---------------------
v_color(BLUE,YELLOW);
v_cls();
```

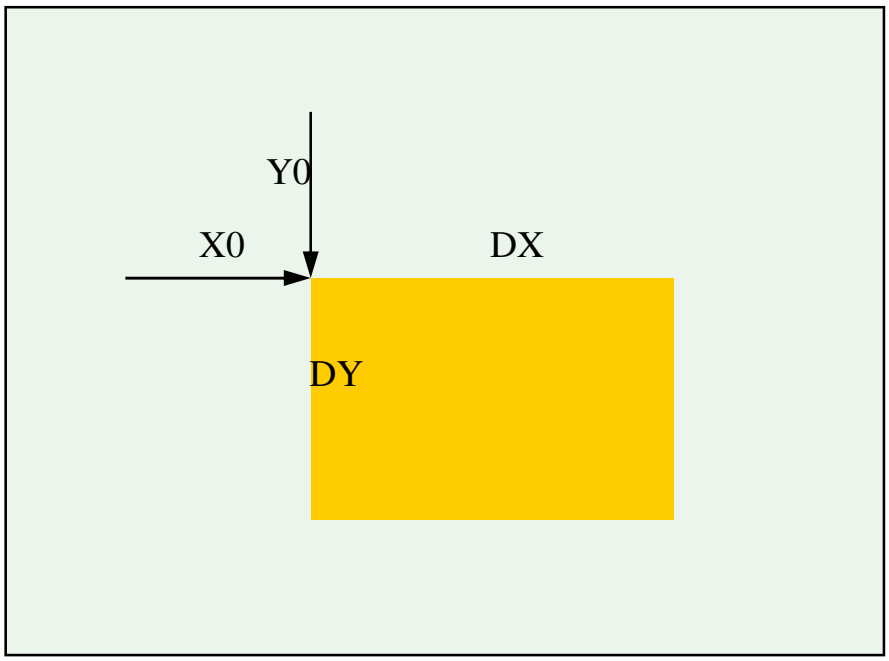


```
void v_line(int x0,int y0,int dx,int dy);
void v_linec(int x0,int y0,int dx,int
dy,int color);
```

It designs a line.

x0 abscissa of the beginning point
y0 ordinate of the beginning point
dx abscissa of the final point less abscissa of the beginning point
dy ordinate of the final point less ordinate of the beginning point
color color of the line (in v_line the color is settato with v_color)

This command designs a line on the screen.



Example:

```
// ----------------------
// in start.c
// ----------------------
v_color(BLUE,YELLOW);
v_cls();
v_linec(10,20,50,80,RED);
while(1);
```

```
void v_rect(int x0,int y0,int dx,int dy);
void v_rectc(int x0,int y0,int dx,int
dy,int color);
```

It designs the contour of an empty rectangle.

x0 abscissa of the beginning point
y0 ordinate of the beginning point
dx width
y0 height
color color of the edge (in v_rect the color is set with v_color)
.

Example:

```
// ---------------------
// in start.c
// ---------------------
v_color(BLUE,YELLOW);
v_cls();
v_rectc(10,20,50,80,RED);
while(1);
```

```
void v_rectfill(int x0,int y0,int dx,int
dy);
void v_rectfillc(int x0,int y0,int dx,int
dy,int color);
```

It designs the contour of an full rectangle.

x0 abscissa of the beginning point
y0 ordinate of the beginning point
dx width
y0 height
color color of the edge (in v_rectfill the color is set with v_color)
.

Example:

```
// ---------------------
// in start.c
// ---------------------
v_color(BLUE,YELLOW);
v_cls();
v_rectc(10,20,50,80,RED);
while(1);
```

```
void v_printchar(unsigned char c);
```

It prints a character

c character to print

It prints a character ascii to the position of the cursor. The cursor comes increasing of the length of the string.

Example:

```
// ---------------------
// in start.c
// ---------------------
v_color(BLUE,YELLOW);
v_cls();
font(off8,1,1);
v_color(TRASP,RED);
v_llocate(2,2);
v_printchar(49);  //it prints the letter A
v_printchar('b'); //it prints the letter B
while(1);
```

```
void v_print(char *s);
```

It prints a string

s string to print

It prints a string that finishes with character ascii 0 to the position of the cursor. The cursor comes increasing of the length of the string.
If the string contains a character ascii 13 (return) the cursor it goes at the beginning to aligned head of the string.

Example:

```
// ----------------------
// in start.c
// ----------------------
v_color(BLUE,YELLOW);
v_cls();
font(off8,1,1);
v_color(TRASP,RED);
v_llocate(2,2);
v_print("prova 1\n");
v_print("prova 2\n");
while(1);
```

```
void lock(void);
void unlock(void);
```

Reservoir and free the writing on screen

The command lock reserves the access to the writing on video to task the current. The command unlock enable the previously classified access with lock.

If more task they write on the video, is necessary that the activities are managed from a semaphore, waves to avoid that the several commands stir themselves, giving place to visualization errors. To such scope the commands of lock can themselves be used and unlock.

Example:

```
// ---------------------
// in task1
// ---------------------
lock();
v_color(BLUE,YELLOW);
font(off8,1,1);
v_llocate(2,2);
v_print("task 1");
unlock();
// ---------------------
// in task2
// ---------------------
lock();
v_color(RED,GREEN);
font(off16,1,1);
v_llocate(5,2);
v_print("task 2");
unlock();
```

KEYBOARDS

The plans developed with Proteus 2007 can turn on hardware various, which:

S400 with touch_screen
S400 with touch_screen and keyboard
S400 with keyboard
ARM with touch_screen
ARM with touch_screen and keyboard
ARM with keyboard
Consul with LCD 2 x 20.
PC

If the objects are used and the compnents standard the program is independent from the hardware, however in cases details (program on consul LCD or I use detail of the keyboard) is possible to approach the keyboard functions directly.
It is necessary to keep in mind that using also a single one of the functions brought back here, the program it will be legacy to a hardware detail and will have to be modified in order to work on other hardware.

```
void v_tik(unsigned char x0);
```
(solo v5-v10-ARM con touch screen)

The display it emits a tick.

x0 mode:
- 0 tick touch screen hardware
- 1 tick touch screen from program
- n>1 it emits a tick of n milliseconds

Example:

```
// in a page…
void Event(bitmap1_onmousedown)()
{
    v_tik(2);
}
```

In Proteus a Bitmap object gives a tick, if touched, only if it changes of image. In the example it forces the tick even if the image remains the same one.

```
void v_spot(unsigned long int leds);
(only S400 consul with keyboard to leds  )
```

Set leds of the keys.

leds configuration on-off of the leds:

Leds is a variable one of 32 bits. To every bit one of the 32 leds is associated a keys of the keyboard. If the corresponding bit is to 1 the led is on, 0 off.

Example:

```
// in COMMON.H…
unsigned long int ledimage=0;

// in a page…
void Event(bitmap1_onmousedown)()
{
   v_spot(ledimage |(1<<14));
}
```

In this example we ligth led the 15 when we touch the object bitmap bitmap1.

```
void keyboard(const char* keyboar_keys);
```
(consul S400 e ARM with keyboard)

Set of the configuration of the keys of the **keyboard hardware**.

Keyboar_keys table of the associated values ascii to the keys.

The keyboard, when pressed, gives a code (scancode) that it characterizes the key that has been pressed. With this function we associate the value ascii that we want to every key.
For the Touch-Screen keyboard there are other functions. For default, the keyboard gives following the 63 scan codes:

code	ascii	code	ascii	code	ascii	code	ascii
1	0	2	1	3	2	4	3
5	4	6	5	7	6	8	7
9	8	10	9	11	F1	12	F2
13	F3	14	F4	15	F5	16	F6
17	F7	18	F8	19	F9	20	F10
21	u (up)	22	d (down)	23	l (left)	24	r (right)
25	x	26	y (yes)	27	n (no)	28	b (back)
29	h	30	-	31	.	32	e (enter)
33	A	34	B	35	C	36	D
37	E	38	F	39	G	40	H
41	I	42	J	43	K	44	L
45	M	46	N	47	O	48	P
49	Q	50	R	51	S	52	T
53	U	54	V	55	W	56	X
57	Y	58	Z	59	+	60	*
61	=	62	/	63	blank		

All the types of keyboard do not have the complete set of keys shown in the brought back table over, but in any case, for the present keys, the code is same for all the keyboards that have the corresponding key.
As an example, the hardware that has only numerical keyboard, will not have some key correspondent to A or B, but the key =, if present, will always correspond to the code of keyboard 61. In such way it is possible to create of the portabile code.

Example:

```
// in COMMON.H…
```

```
const char tast1[]={0,
      '0','1','2','3','4','5','6','7','8','9',
      200,201,202,203,204,205,206,207,208,209,
      'u','d','l','r','x','y','n','b','h','-',
      '.','e','A','B','C','D','E','F','G','H',
      'I','J','K','L','M','N','O','P','Q','R',
      'S','T','U','V','W','X','Y','Z','+','*',
      '=','/',' '};

// in start.c…
keyboard(tast1);
```

In this example we associate the keyboard standard to the values standard of the keys.

```
void keyboard(const char* keyboar_keys);
```
(v5/v10/ARM with touch screen)

Set of the configuration of the keys of the **keyboard on the touch screen**.

Keyboar_keys table of the associated values ascii to the keys.

In the event of keyboard on touch screen, the function keyboard declares the characters to associate from above to the keyboard, scan from left towards right and towards the low. In this case a mapping does not exist standard for the keys being these defined from the commando **keyon**.

For ulterior explanations and examples one sends back the commandos **keyon and keyshow**.

```
void key_show(int lin,int col,int nx,int
ny,char *show_keys,const char* keyboar);

void key_msg(char *msg);

void keyoff(void);
```
(solo v5/v10/ARM with touch screen)

key_show Enable the visualization of the keyboard on the touch screen.
key_msg It show a message on the display of the keyboard as son enable from key_show.
keyoff ago to disappear the keyboard from the screen.

lin line of beginning of the keyboard (angle up on the left).
col column of beginning of the keyboard (angle up on the left).
nx number of keys in horizontal.
ny number of keys in vertical.
show_keys string of the characters to write on the keys (from left towards right, from above towards the low) **keyboar** table of the associated values ascii to the keys.
msg it is string ascii of the message to visualize.

Key_show extension the keyboard over the image, defines the characters to over writer to the keys and the characters ascii correspondents. The dimensions in pixel of the keys depend from the resolution of the monitor. Key_msg extension a message that generally explains this that it attends to us that comes push on the keyboard. Such message to disable as soon as a key is push.
At the end, keyoff ago to disappear the keyboard, restoring the below objects.
These functions are calls automatically from the Edit objects, and come only used in cases details.
Where possible, to avoid to use them directly, because they are not compatible with version PC.

These functions always come used jointly, and in the order over wiew.

Example:

```
void bitmap1_onmouseup()
{
   const  char keyboard_1[]={"1234567890.-uzxe"};
   static char strinput[8];
   long int key=0;
   key_show(2,2,4,4,"1234567890.-u xe",keyboard_1);
   key_msg("Password ?");
   t_input(strinput,8);
   key=atol(strinput);
   if ((key==12345) || (key==23456))
   {
     page_end=1;
     page_num=_page1;
   }
   keyoff();
}
```

```
void key_wait(void);

void key_release(void);
```
(consul with keyboard)

They attend respective the pressure and the release of a key.

```
unsigned char t_inke(void);
unsigned char t_inkey(void);
unsigned char t_inkey(char mode);
```

These two functions give the value ascii of the pressed key. The ferst form (**t_inke**) gives the **instantaneous** value of the pressed key. The second form **(t_inkey)** is equivalent to t_inke in the S400 version.

In version ARM t_inkey it has the argument mode:

If mode=0 or if the hardware has the keyboard, it is equivalent to t_inke.

If the hardware has the touch screen, visualizes a keyboard, whose layout it depends from the value of mode:

mode= **_LOGIC_** keyboard Yes/No/Esc
mode= **_FUNCTION_** function keyboard-F5/Esc
mode= **_NUMBER_** number keyboard
mode= **_ALPHA_** alphanumeric keyboard

The keyboard go off automatically after that a key has been pressed.

Example:

```
void Event(onentry)()
{
   key=t_inke();
   if (tasto==202)
   {
      page=_page2;
      page_end=1;
   }
   if (key==203)
   {
      pagina=3;
      page_end=1;
   }
}
```

```
void t_input(char * string,short int
max);
```

string address of the string to fill up
max the maximum number of characters to read.

This function reads from keyboard a string of characters and it puts it in the variable one specified.

Example:

```
void bitmap1_onmouseup()
{
   const  char keyboard_1[]={"1234567890.-uzxe"};
   static char strinput[8];
   long int key=0;
   key_show(2,2,4,4,"1234567890.-u xe",keyboard_1);
   key_msg("Password ?");
   t_input(strinput,8);
   key=atol(strinput);
   if ((key==12345) || (ke==23456))
   {
     page_end=1;
     page_num=_page1;
   }
   keyoff();
}
```

SERIAL PORT

The PLC S400 has 4 asynchronous serial ports

COM1	10	…	625000 baud
COM2	4800	…	625000 baud
COM3	57600	…	5000000 baud
COM4	57600	…	5000000 baud

The baud rates is defined from the relationship:

BAUD_RATE_MAX/N

with entire N

If the PLC has a consul (monitor or lcd) port COM3 it is used from the operating system for the communication with the consul. For equipment without display COM3 he is usable from the program.

The PLC with microprocessor ARM have 2 serial port asynchronous:

COM1	70	…	4500000 baud
COM2	70	…	4500000 baud

It is possible for the PLC of this series to use **any baud rates**, arranging the UART of prescaler fractional.

The programmer can use the communication via seriale at the level of single character (way 0), of blocks of data (syel mode 1 and 2), or defining an own protocol of communication.

In way 0 (default) the operating system manages the coms to interrupt with of buffers the circulars fifo of 256 bytes.

In syel mode 1 they come managed long messages to maximum 256 bytes. If an own protocol is defined, the received data comes passed directly to the subroutine defined from the programmer.

In syel mode 2 they come managed messages of the definable maximum length. Protocol syel mode 2 is started by means of the commando pclib_open.

pclib_open(COM *com,int baudrate,int max_buffer_len)

It is necessary to specify the number of centers them (the plc same) and of peripheral (the remote equipment, generally a PC)

To this point it is possible to define buffers of the messages of transmission and the reception.

Example:

```
pclib_open(COM2,625000,4096);
COM2->centr=1;
COM2->perif=15;
_rx(COM2,1,(char *)&buffr1,sizeof(buffr1),subrx1);
_rx(COM2,2,(char *)&buffr2,sizeof(buffr2),subrx2);
_tx(COM2,1,(char *)&bufft1,sizeof(bufft1),subtx1);
_tx(COM2,2,(char *)&bufft2,sizeof(bufft2),subtx2);
```

Each COM it is associated to a structure

```
typedef struct COM
{
  unsigned char open;      //0   flaag COM open
  unsigned char mode;      //1   protocol type
  long int baud;           //2   baud rate
  void(*subrx)();          //6   address subroutine associated a rx
  void(*subtx)();          //10  address subroutine associateda tx
  unsigned int  rx0;       //14  address buffer circular start rx
  unsigned int  rx1;       //16  address buffer circular end rx
  unsigned int  tx0;       //18  address buffer circular start tx
  unsigned int  tx1;       //20  address buffer circular end tx
  unsigned char *rxbuff;   //22  buffer of rx
  unsigned char *txbuff;   //26  buffer of tx
  unsigned char rxstatus; //30  status rx
  unsigned char txstatus; //31  status tx
  unsigned char rxchk;     //32  checksum rx
  unsigned char txchk;     //33  checksun tx
  unsigned int  rxsize;    //34  n. characters rx
  unsigned int  txsize;    //36  n. characters waiting.
  unsigned char centr;     //38  n. master
  unsigned char perif;     //39  n. slave
  unsigned char echo;      //40  1= disable rx Durino tx.
  unsigned char txrun;     //41  1=tx is running
  void(*protocol)(struct COM*,int); //42 custom interrupt subroutine
}COM;
```

Generally the programmer does not approach directly the data of this structure, except if own protocol uses its.

Example:

This is my protocol, that it comes called for every data received, and comes passed the address to it of com and the received data.
In the example increment the counter pippo every time that I receive a character ascii A from COM2 and answers with character C every time that receives a character B.

In common.h

```
int pippo=0;
void com_int(COM * this_com,int data)
{
   if (data=='A') pippo++;
   if (data=='B') com_tx(this_com,'C');
}
```

In start.c

```
com_open(COM2,57600);    //I open COM2 a 57600 baud
com_disable(COM2);       //disable
COM2->protocol=com_int; //set to my protocol
com_enable(COM2);        //enable
```

In the example, it is made approached the structure of the COM directly in order to assign the address of the ruotine of management of the protocol.
The protocol thus defined, that it comes executed at the level of ruotine of interrupt, is that one of reception. The transmission protocol is in any case a function that sends the data by means of of the instructions com_tx (COMn, data).

```
int  com_open(struct COM *porta,long int
baud);
```

Open a COM.

porta name of the serial port (COM1,COM2,COM3 o COM4)
baud baud rate (the baud rate min. e max. are fuction of tue com)

COM1	min. 10	max. 625000 baud
COM2	min. 4800	max. 625000 baud
COM3	min. 57600	max. 5000000 baud
COM4	min. 57600	max. 5000000 baud

The baud installments maximum moreover is limited from the type of interface (RS-232, fiber optic.) and from the length of the connection. For further information to care to see the detailed lists of the PLC S400.

Example:

In start.c

```
com_open(COM1,57600);     //apen COM1 a 57600 baud
com_open(COM2,625000);    //apen COM2 a 625 KBaud
```

```
int  com_close(struct COM *porta);
```

Close a COM.

porta name of serial port (COM1,COM2,COM3 o COM4)

Generally, once opened a com, not there is some reason in order to close it, except if we want to reopen it with various modality.

Example:

```
com_close(COM1);     //close COM1
com_close(COM2);     //close COM2
```

```
int   com_enable(struct COM *com);
int   com_disable(struct COM *com);
```

Enable and disabile a COM.

porta name of serial port (COM1,COM2,COM3 o COM4)

A serial port, once opened, is automatically enabled.
In case we must change of directly the attributes, going to modify the associated structure, is good norm to disable the COM during the access to the relative structure, otherwise could be had turned out unforeseeable if receipts of the characters come during the modification of the parameters.

Example:

In start.c

```
com_open(COM2,57600);    //open COM2 a 57600 baud
com_disable(COM2);       //disable
COM2->protocol=com_int; //my setting
com_enable(COM2);        //enabled
```

```
int  protocol_mode(struct COM *porta,char mode);
```

It assigns a protocol to the COM.

porta name of serial port (COM1,COM2,COM3 o COM4)
mode tipe of protocol.

Currently three types of protocol are available:

0 protocol at the level of single character, with buffer of reception (it is the way of default at the opening of the COM).
1 protocol at the level of block (Syel protocol block mode 1)
2 protocol at the level of functions (Syel protocol block mode 2), than schism but by means of the command **pclib_open.**

If the protocol mode it is 1 it is possible to assign a ruotine of rx with the command on_rx that it will come called once received the block of data, and a ruotine of tx with the command on_tx, that will come called before the shipment of the data.

Example:

In start.c

```
com_open(COM2,57600);    //open COM2 a 57600 baud
protocol_mode(COM2,1);   //block mode 1
on_rx(COM2,my_rx_sub);   //start subroutine of rx
+
```

For moreelucidations care protocol mode 2 you see the example modo2_pclib in directory the ESEMPI_ARM

```
int  com_speed(struct COM *porta,long int
baud);
```

It changes the baud-rate of an open COM.

porta name of serial port (COM1,COM2,COM3 o COM4)
baud neuw baud rate.

A com, once opened, works to the baud rate specified with the opening command. With the command com_speed it is possible to change to such baud rate to the flight, without to close neither to disable the COM. If there is a character in reception course, this will be lost.

Example:

In start.c

```
com_open(COM2,57600);   //open COM2 to 57600 baud
com_speed(COM2,625000); //it change the baud rate to
625K
```

```
int   com_tx_empty(struct COM *port);
```

1 returns if the buffer of transmission it is empty (only protocol mode 0)

```
int   com_tx_full(struct COM *port);
```

1 returns if the buffer of transmission it is full (only protocol mode 0)

```
int   com_tx_size(struct COM *port);
```

It returns with the number of characters still available in the buffer of transmission (only protocol mode 0)

```
int   com_rx_empty(struct COM *port);
```

1 returns if the buffer of reception it is empty (only protocol mode 0)

```
int   com_rx_size(struct COM *port);
```

It returns with the number of characters still not read in the buffer of reception (only protocol mode 0)

```
int   rxchars(struct COM *port);
```

It returns with the number of characters receipts (only protocol mode 0)

Example:

```
com_open(COM2,57600);   //apro COM2 a 57600 baud
// ...
While(!com_rx_empty(COM2))
{
   msg[address++]=com_rx(COM2);
}
```

```
int  com_tx(struct COM *port,unsigned
char c);
```

It sends a character through a COM (only protocol mode 0).

Port name of seriale port (COM1, COM2, COM3 or COM4)
c character to send.

Example:

```
com_open(COM2,57600);    //open COM2 to 57600 baud
com_tx(COM2,'A');        //send the character A
```

```
int   com_rx(struct COM *port);
```

it receives a character through a COM (only protocol mode 0).

port name of serial port (COM1,COM2,COM3 o COM4)

Example:

```
com_open(COM2,57600);   //open COM2 to 57600 baud
// ...
While(!com_rx_empty(COM2))
{
   msg[address++]=com_rx(COM2);
}
```

```
int  com_txs(struct COM *port,unsigned
char *c);
int  com_txsl(struct COM *port,unsigned
char *c,int len);
```

It sends a string of characters through a COM (only protocol mode 0).

Port name of serial port (COM1, COM2, COM3 or COM4)
c address to the string of characters to send.
len number of characters.

com_txs it sends to a string of characters **till the first null character** of the string (modality text).
com_txsl it sends to **the specific number of characters**, comprised the null characters (modality binary).

Example:

```
char msg1[16];
Char msg2[]={0,0,0,3,32,0x44,0x55,3};
com_open(COM2,57600);   //open COM2 to 57600 baud
strcpy(msg1,"Ciao !\n");
com_txs(COM2,msg1);     //send message 1
com_txsl(COM2,msg2,8);  //send message 2
```

```
int  bload(struct COM *port,void
*buffer,int len);
```

It reads a block of data

Port name of serial port (COM1, COM2, COM3 or COM4)
Buffer string in which copying the message.
Len number of the maximum characters to read.

This command, valid only in **BLOCK MODE 1**, reads the message received on the specified Com, restoring the same Com to receive new messages.
If no message has arrived, returns with code of error **- 1**.
If there is a valid message, returns with code of error **0**.
Normally this command finds in the ruotine of on_rx of the Com, in such case is sure that he will come only called if it is a message to read.
If the length of the message exceeds the length specified in the field len, first len bytes of the message they will only come copied in the buffer.
A message is considered valid if it has the just one header, if it is destined to us (id=COMx-> centr) and if the checksum is corrected.
For the calculation of the checksum the entire message is taken in consideration and not only the number of bytes that we want to read.

Example:

In common.h

```
char msg1[16];
void subrx2(void)
{
   bload(COM2,msg1,16);
}
```

In start.c

```
com_open(COM2,57600);
protocol_mode(COM2,1);
COM2->centr=4;
onrx(COM2,subrx2);
```

```
int  bsave(struct COM *port,void
*buffer,int len);
```

It sends a block of data

door name of serial port (COM1, COM2, COM3 or COM4)
buffer string in which the message is found.
len number of characters to send.

This command, valid only in **BLOCK MODE 1**, Sends the message on the specified Com, to the peripheral one whose address is specified from COMx-> perif.

Example:

```
char buffer1[16];
int num=1;
com_open(COM2,57600);
protocol_mode(COM2,1);
COM2->centr=4;
COM2->perif=12;
onrx(COM2,subrx2);
sprintf(buffer1,"MESSAGE %d",num++);
bsave(COM2,buffer1,strlen(buffer1));
```

```
int  com_rxlen(struct COM *port);
```

It returns with the number of characters receives (only protocol mode 1)

```
int  com_rxcode(struct COM *port);
```

It returns with the first received character (only protocol mode 1)

Example:

In common.h

```
char msg1[16];
char msg2[16];
void subrx2(void)
{
   if (com_rxcode(COM2)==1) bload(COM2,msg1,16);
   else bload(COM2,msg2,16);
}
```

In start.c

```
com_open(COM2,57600);
protocol_mode(COM2,1);
COM2->centr=4;
onrx(COM2,subrx2);
```

```
int  onrx(struct COM *port,void
(*subroutine)());

int  ontx(struct COM *port,void
(*subroutine)());
```

It associates a subroutine to execute when is received a package in mode 1 (onrx) or before transmitting a package in mode 1 (ontx).

Example:

In common.h

```
char msg1[16];
char msg2[16];
void subrx2(void)
{
   if (com_rxcode(COM2)==1) bload(COM2,msg1,16);
   else bload(COM2,msg2,16);
}
```

In start.c

```
com_open(COM2,57600);
protocol_mode(COM2,1);
COM2->centr=4;
onrx(COM2,subrx2);
```

SERIAL FLASH

The PLC of the S400 series have a serial memory not flown them flash of variable dimension according to the model from 2 Mbytes to 8 Mbytes. The first 1,5 Mbytes of the flash are used for the Dos and the memorization of the job program.
The successive part is on hand of the programmer, that it can use in order to write and to read the parameters of the machine, the programs of customer or other. These operations come executed by means of the commands described here.

The PLC of series ARM have memory card SD of 2 Gigabytes, we can approach to the SD by means of functions standard C for management of file system (fopen, fread, fprintf, fscanf etc).
For compatibility with the S400 the functions of serial flash are implemented, and use like simulated physical support a flash from the C:\FLASH.BIN file, that it comes created automatically, if not present, to the first access in writing to the simulated flash.

```
int sf_wdata(unsigned char *src,long int
dst,long int num);
```

It writes data in flash

src point to the structure or the carrier of data to write in flash
Dst address of flash beginning from which to write the data.
Num number of bytes writing.

This command sees only the area of memory flash to use to the programmaer, removing automatically the offset one of 1.5 Mbytes. The first address of valid departure is 0.
The operation has only succeeded if the area of memory in which we want to write is entire contained in the flash.
The return code is the checsum to 8 bit of the saved data.

Example:

In common.h

```
struct
{
   char name[40];
   int  number;
} my_record;
```

elsewhere ...

```
sf_wdata(&my_record,0,sizeof(my_record));
```

```
int sf_rdata(unsigned char *src,long int
dst,long int num);
```

It reads data from the flash

src point to the structure or the carrier of data to read from flash
Dst address of flash beginning from which to read the data.
Num number of bytes reading.

This command sees only the area of memory flash of use to the programmer, removing automatically the offset one of 1.5 Mbytes.
The first address of valid departure is 0.
The operation has only succeeded if the area of memory in which we want to read is entire contained in the flash.
The return code is the checsum to 8 bit of the read data.

Example:

In common.h

```
struct
{
   char nome[40];
   int  numero;
} my_record;
```

elsewhere ...

```
sf_rdata(&my_record,0,sizeof(my_record));
```

```
int sf_cdata(unsigned char *src,long int
dst,long int num);
```

Verification of the data in flash

Src point to the structure or the carrier of data to compare with the flash
dst address of flash beginning from which to confront the data.
Num number of bytes confronting.

This command sees only the area of memory flash to use to the programmer, removing automatically the offset one of 1.5 Mbytes.
The first address of valid departure is 0.
The operation has only succeeded if the area of memory in which we want to verify is entire contained in the flash.
The return code gives the number of errors, that is the number of the bytes that the verification has found various.

Example:

In common.h

```
struct
{
   char nome[40];
   int  numero;
} my_record;
```

elsewhere ...

```
sf_cdata(&my_record,0,sizeof(my_record));
```

```
int sf_wcode(unsigned char *src,long int
dst,long int num);
int sf_rcode(unsigned char *src,long int
dst,long int num);
int sf_ccode(unsigned char *src,long int
dst,long int num);
```

Like the commands seen over, but reported to the part of the flash classified to the Dos.
Not to use never such commandos, pain the destruction of firmware of the PLC.

PEN USB (only S400 version)

In version ARM pen USB is seen and managed like VOLUME D: with the normal commands of the file system (fopen, fclose, fread etc) and know file-systems FAT12, FAT16 and FAT32, having approached the under-directory of any level.

It is possible to connect to the PLC of the S400 series an interface USB in a position to directly reading to the content of a portable memory of type Pen USB or, (with opportune adapter USB), a Compact Flash or a memory SD.
The scope is that one of being able to read or to write of the files compatible with the PC, in order to exchange data with this last one.
A complete management of file system FAT is outside from the scopes and the possibilities of memory of the PLC, however it is possible to read and to write files with the following limitations:

-The support must be formatted in modality **FAT16**
- It can be only approached the **first partition**
- Such partition does not have to exceed the **2 Gbytes**
- The files they must be found in the **directory main** (root)
- It is possible to open **single file for time**.
- To only hold on pen USB the files destined to the exchange with the PLC.

Two series of commands exist in order to approach pen USB:

- a first set of dedicated commands (cf_init, cf_dir etc.)
- a second set standard C (fopen, fread, fwrite etc.)

The set standard C is complete and allows to make all the operations standard on the files.
The dedicated set contains a repetition of the commands of the complete set, plus some commands who do not exist in the C standard, (like to es. the command for the directory of the files or in order seeing if the pen is inserted). While the set dedicated ago reference exclusively to pen USB, the set standard can be used also in order to approach to other peripheral ones of mass connected to the PLC, like the hard disk.

The dedicated commands return with an error code, than if it is zero indicate the happened one of the operation, otherwise they indicate the reason of the failure.

He follows a list of the possible codes of error:

```
;  error code:
;
;  0   O.K.
;  1   I/O GENERAL POURPOSE ERROR
;  2   CARD NON INSERTED
;  3   FILE ALREADY OPEN
;  4   FILE NOT FOUND
;  5   DISK FULL
;  6   FILE EXISTS
;  7   DIRECTORY FULL
;  8   FILE NOT OPEN
;  9   NUMBER WRONG PACKAGE
; 10   TIMEOUT
; 99   END OF DIRECTORY
```

Being able interface USB to manage single file for time, it is well that the several ones task approach to you with a semaphore.
To such scope the instructions exist **mount** and **unmount**.

```
void cf_init(void);
```

Set USB interface

This command is for internal use only of the operating system and it is not never necessary to use it in the program directly.

```
int  cf_dir(char *filename,char
*result,char mode);
```

Verification of the data in flash

filename name of the file (can contain the character *)
result point to a string in which writing the complete name of the file found.
mode mode of access to the command.

This command tries in directory main file whose name corresponds to the same like first argument.
If mode=0 he tries the first occurrence of the file.
If mode=1 he tries the successive occurrence.
He returns with the error code.

Example:

```
char dir[100][25];      //string of the results

int directory(char *filetype)
{
   static char s[128];  // temporary string
   int er=0;            // clear error code
   int first=0;         // set to start directory
   int n=0;             // clear num. Files founded
   while(!er)
   {
      er=fdir(filetype,s,first);
      first=1;          // next file
      if(!er)           // if file founded ...
      {
         strncpy(dir[n],s,23);
         dir[n++][23]=0;
         if(n>99) break;
      }
   }
   return n;
}

x=directory("*.dat");
```

```
int  cf_open(char *filename,char mode);
```

Open a file

filename name of the file
mode mode opening.

This command tries in directory main file whose name corresponds to the same past like first argument, opens therefore it in the modality specified from the second parameter

Mode

1	read
2	write
3	create
4	random r/w
5	append

Return with the error code.

The name of the file must be compatible DOS (in characters capital letters, maximum 8 characters + extension max 3 characters)

Example:

```
cf_open(”FILE1.DAT”,1);
```

that standard C is equivalent to the form:

```
FILE *f;
f=fopen(”FILE1.DAT”,”r”);
```

```
int  cf_test(void);
```

Test the presence of pen USB, closes files opened.

This command interrogates pen USB in order to verify of the presence. It closes moreover eventual files opened and returns with the error code.

Example:

```
int err;
if(err=cf_test()) printf("error %d",err);
```

```
int  cf_close(void);
```

It closes the file opened on pen USB.

This command closes the rows opened on pen USB.
In present the version the operating system concurs of opens single file for time, for which, before opening new file, it is necessary to close the precedence.
If no file were open, this command does not make nothing and he does not mark it error.
For precaution it is to give a command of closing files at the beginning of the program always well, as the management of the files happens in interface USB and not in the PLC, for which an eventual Reset of the equipment it can leave open of the files on the interface.

It is equivalent to the command fclose (FILE *f).

Example:

```
cf_open(”FILE1.DAT”,1);
. . . .
cf_close();
```

that standard C is equivalent to the form:

```
FILE *f;
f=fopen(”FILE1.DAT”,”r”);
. . . .
fclose(f);
```

```
int  cf_del(char *filename);
```

It clears more or files on pen USB.

filename name of the file (the character can contain *)

This command cancels all the files whose name corresponds to the string used like parameter.

He does not have equivalents in the standard library C.

Example:

```
cf_del("*.*");
```

Clear all files on the pen USB.

```
int  cf_rb(char *buffer,int len);
```

It reads binary data from open file

buffer vector to put reading data .
len number of bytes to read

This command reads the specific number of bytes from the file running, beginning from the running position, and increases the position.
At the opening the running position is the beginning of the file (opening in mode read or random).
In order to change the running position to use the command cf_seek.

It is equivalent to the command fread of the C standard.

Example:

```
cf_open(”FILE1.DAT”,1);
cf_seek(1200);
cf_read(my_buffer,200);
cf_close();
```

It is equivalent to the command standard C :

```
FILE *f;
f=fopen(”FILE1.DAT”,”rb”);
fseek(1200,SEEK_SET);
fread(my_buffer,200,1,f);
fclose(f);
```

In this example 200 bytes are read beginning from the byte n. 1200 of file FILE1.DAT

```
int  cf_wb(char *buffer,int len);
```

It writes binary data in the open file.

buffer vector where are data to writer.
len number of bytes to writer

This command writes the specific number of bytes in the file running, beginning from the running position, and increases the position.
At the opening the running position is the beginning of the file (opening in write mode, create or random). , or the end of the file (opening in mode append).
In order to change the running position to use the command cf_seek.

It is equivalent to the fwrite command of the C standard.

Example:

```
cf_open(”FILE1.DAT”,4);
cf_seek(1200);
cf_write(my_buffer,200);
cf_close();
```

It is equivalent to the command standard C :

```
FILE *f;
f=fopen(”FILE1.DAT”,”wb”);
fseek(1200,SEEK_SET);
fwrite(my_buffer,200,1,f);
fclose(f);
```

In this example 200 bytes are writed beginning from the byte n. 1200 of file FILE1.DAT

```
int  cf_seek(long int posiz);
```

Set the position current of reading or writing in the file open.

posiz current position (relative to thebeginning of the file)

This command changes to the position reading-writing current.
If we go beyond the end of the file, in writing case, the file he will come filled up of characters zero till the running position.

It is equivalent to the fseek command of the C standard.

Example:

```
cf_open(”FILE1.DAT”,1);
cf_seek(1200);
cf_read(my_buffer,200);
cf_close();
```

It is equivalent to the command standard C :

```
FILE *f;
f=fopen(”FILE1.DAT”,”rb”);
fseek(1200,SEEK_SET);
fread(my_buffer,200,1,f);
fclose(f);
```

In this example 200 bytes are readed beginning from the byte n. 1200 of file FILE1.DAT

```
int  cf_rd(char *buffer);
```

It reads a string of data from the file of open text.

buffer vector in which putting the read data.

This command reads a string of bytes from the file running, beginning from the running position, and increases the position.
He stops himself if he finds a **character CR** (ascii 13) or the **end of file**, in any case **after 240 bytes** read .
At the opening the running position is the beginning of the rfile (opening in mode read).

It is equivalent to the fgets command of the C standard.

Example:

```
cf_open(”FILE1.DAT”,1);
cf_read(my_record);
cf_close();
```

It is equivalent to the command standard C :

```
FILE *f;
f=fopen(”FILE1.DAT”,”r”);
fgets(my_record,240,1,f);
fclose(f);
```

In this example the first line of the file of text FILE1.DAT is read.

```
int  cf_wr(char *buffer);
```

A line of text in the file open is writes.

buffer vector containing the text.

This command writes a line of text in the file opened, finishing it with a character ascii CR (13), and increases the position.
At the opening the running position is the beginning of the file (opening in mode write or created) or the end (opening in mode append).

It is equivalent to the fputs command of the C standard.

Example:

```
cf_open(”FILE1.DAT”,2);
cf_wr(my_buffer);
cf_close();
```

It is equivalent to the command standard C

```
FILE *f;
f=fopen(”FILE1.DAT”,”w”);
fputs(my_buffer,f);
fclose(f);
```

In this example one opens the file and a text line is written.

```
int   cf_status(int mode);
```

It reads the status of the file

It refresh the variables:

fl_eof (1 if eof)
fl_pointer (current position on the file)
fl_size (size of the file)

mode command:

mode=0: return with eof+ actual_byte pointer + file len
mode=1: return with eof+ disk capacity
mode=2: return with eof+ disk capacity + disk free

FILES MANAGEMENT

The version current of Proteus previews pen USB like only accessible support of data with the functions of file system standard of language C on the PLC of the S400 series.
Interface USB of the S400 supports single a FCB, for which it is possible to open single file for time, in the directory main of pen USB, that it must be formatted in mode FAT16.

Obviously these limitations do not exist if we develop with Proteus a program that turns on PLC ARM or PC. In order to in the event guarantee the exclusive access to the file system of more task, they have been implemented the functions of semaphore:

```
void mount(void);
void unmount(void);
```

that they allow to create a mechanism of mutual exclusion during the access to the file system.

Example:

```
FILE *f;                    //set the FCB
```

In a task:

```
mount();                    //enable FCB
f=fopen("FILE1.DAT","w");//open the file
fputs(my_buffer1,f);        //write
fclose(f);                  //close the file
unmount();                  //disabile FCB
```

In a other task:

```
mount();                    //enable FCB
f=fopen("FILE2.DAT","r");//open the file
fgets(my_buffer2,f);           //read
fclose(f);                  //close the file
unmount();                  //disabile FCB
```

Le funzioni standard C per la gestione dei files sono:

```
FILE      *fopen(const char *name,const char *mode);
int       fclose(FILE *file);
size_t    fread(void *buffer,size_t count,size_t
          num,FILE *file);
int       fgetc(FILE *file);
char      *fgets(char *buffer,int max,FILE *file);
int       fscanf(FILE *file,const char *format,...);
size_t    fwrite(const void *buffer,size_t
          count,size_t num,FILE *file);
int       fputc(int c,FILE *file);
int       fputs(const char *buffer,FILE *file);
int       fprintf(FILE *file,const char *format,...);
int       fseek(FILE *file,long offset,int whence);
long int  ftell(FILE *file);
int       feof(FILE *file);
long int  flen(FILE *file);
long int  fpointer(FILE *file);
```

and **limitedly to the PLC of series ARM**:

```
void     mount(void);  mount the file system standard
void     unmount(void); flush file system standard
int      ffind(const char *filename,char *result,char
         mode);
void     set_current_drive(const char *drivename);
int      finit (void);
int      fdelete (const char *filename);
int      frename (const char *oldname, const char
         *newname);
int      fcopy (const char *oldname, const char
         *newname);
U32      ffree (const char *drive);
int      fformat (const char *drive,U32 size);
int      mkdir (const char *dirname);
int       remdir  (const char *dirname);
void     chgdir (const char *dirname);
void     set_current_dir (const char *dirname);
char     *get_current_dir(void);
```

For the description of such functions is sent back to the manual of the functions standar C, section STDIO.H

SERIAL EXPANSION I/O

The PLC S400 supports the expansion of the I/O by means of modules of serial interface in RS-232 or fiber optic.
It is possible to connect between they in cascade till 16 of such modules, everyone of which can have till 128 inputs, or till 128 outputs, or both.
So that such modules are managed correctly from the system, it is necessary that they are declared (normally in start.c) by means of the command:

```
void refresh(unsigned int perif,COM
*com,long int baud,int numin,int
firstin,int numout,int firstout,int
timeout);
```

perif number of the periferic (0..15)
com serial port were the module is connecting (COM1...COM4)
baud baud rate (normaly 125000)
numin number of inputs in the module (0..128)
firstin first input to associate the module (>32)
numout number of outputs in the modulo (0..128)
firstout fierst out to associate the module (>32)
timeout timeout max in mSec. (normaly 20)

Limitations:

The COM3 is reserved to the monitor if the PLC has one.
Modules on the same serial port must have same the baud-rate.
The inputs of various modules do not have place upon

Example:

```
refresh(0,COM1,125000,64,33,0,0,20);
refresh(1,COM1,125000,0,0,64,33,20);
refresh(2,COM1,125000,64,97,64,97,20);
```

EXPANSION CAN

The PLC S400 supports completely in transparent mode to the programmer till 6 modules of expansion CAN, everyone of which adds to the I/O of the PLC:

2 axis by encoders up-dn with zero ref
2 analog out
4 analog in
16 digital input
16 digital out

In this modular structure the PLC implements the first two modules (module 0 and 1) and the successive ones are constituted from the expansions.

Module	2	3	4	5	6	7
Encoders	5-6	7-8	9-10	11-12	13-14	15-16
Anal.Out	5-6	7-8	9-10	11-12	13-14	15-16
Anal.In	9-12	13-16	17-20	21-24	25-28	29-32
Inputs	i32-47	i48-63	i64-79	i80-95	i96-111	i112-127
Outputs	o32-47	o48-63	o64-79	o80-95	o96-111	o112-127

These modules are thought in order to expand the axis managed from the PLC but they can be used also for other scopes.
If it is necessary to connect more than 6 modules than expansion, this is possible managing the same ones as peripheral CAN-OPEN.

So that the PLC manages the expansions correctly, it is necessary to declare them by means of the command:

```
void can_perif(int n,int t,int baud);
```

n number of remote modules (from 0 to 6)
t time of refresh in millisecond (normaly 1)
baud baud rate of the CAN in Kbaud (normaly 1000)

Example:

```
can_perif(2,1,1000);  //2 remote modules
```

For the controllers of series ARM exists an alternative form of this command:

```
void can_perif_ext(int n,int t,int
baud,int first);
;
```

n numbers of external module (from 0 to 6)
t time of refresh in milliseconds (normally 1)
baud baud rate of CAN in Kbaud (normally 1000)
first first external module.
If equal to zero the first module of expansion is associated to i1-i16, o1-o16, encoder1-encoder2, anal1-anal2, pot1-pot4.
If equal to 1, the first module of expansion is associated to i17-i32, o17-o32, encoder3-encoder4, anal3-anal4, pot5-pot8.
If equal to 2 it is equivalent to the commando can_perif.

This form of the command is useful above all in the event in which the program turns on an **unit without physical I/O**, for which the physical support of the I/O remits to a peripheral external.

Example:

```
can_perif_ext(2,1,1000,0);  //2 module of expansion
with start from i1, o1, encoder1, pot1, anal1
```

CAN-BASIC

The compiler Proteus ES has a series of functions that allow to manage the peripheral CAN to low level.

A message CAN is sent and received with the structure:

```
typedef struct
{
  unsigned int Frame; // Bits 16..19: DLC - Data Length Counter
                      // Bit 30: Set if this is a RTR message
                      // Bit 31: Set if this is a 29-bit ID message
  unsigned int MsgID; // CAN Message ID (11-bit or 29-bit)
  unsigned int DatA;  // CAN Message Data Bytes 0-3
  unsigned int DatB;  // CAN Message Data Bytes 4-7
} CANBASE_MSG;
```

The relative commandos to the management of the CAN in base way are:

For the first CAN device:
short **canbase_init**(unsigned short can_isrvect,unsigned int can_btr);
short **canbase_txmsg**(CANBASE_MSG *pTransmitBuf);
short **canbase_rxmsg**(CANBASE_MSG *pReceiveBuf);

Per il secondo device CAN (dove presente):
short **canbase_init_2**(unsigned short can_isrvect,unsigned int can_btr);
short **canbase_txmsg_2**(CANBASE_MSG *pTransmitBuf);
short **canbase_rxmsg_2**(CANBASE_MSG *pReceiveBuf);

short **canbase_init**(unsigned short can_isrvect,unsigned int can_btr);
short **canbase_init_2**(unsigned short can_isrvect,unsigned int can_btr);

can_isrvect it is the priority of management of CAN messages . This parameter is present for compatibility with other versions of PROTEUS, it isn't influential, generally and it comes assigned value to it 4.

can_btr it is the baud rate of CAN. It can to assume the values:
CANBitrate125k
CANBitrate250k
CANBitrate500k
CANBitrate1M

This command initializes the CAN peripheral .
If we want to use a baud rate various installments from those specifying in the list of the values of can_btr, we can at any moment change to the baud installments with the command:

can_freq(U32 freq); per il CAN1
can_freq_2(U32 freq); per il CAN2

Where **frequency** may be any numeric value.

short **canbase_txmsg**(CANBASE_MSG *pTransmitBuf);
short **canbase_txmsg_2**(CANBASE_MSG *pTransmitBuf);

Trasmette il messaggio CAN contenuto nella struttura passata come parametro, secondo le specifiche della struttura stessa.
Ritorna con codice ZERO se i buffers di trasmissione sono pieni o in caso di BUSOFF (bus del can interrotto)
Altrimenti ritorna con un valore diverso da 0.

It sends the message CAN contained in the structure passed as a parameter, according to the specifications of the structure.
Return code zero if the transmission buffers are
full or if BUSOFF (CAN bus stop)
Otherwise returns a value other than 0.

Example:

```
CANBASE_MSG TXBuf;              // Buffer  tx CAN message
canbase_init(4,CANBitrate1M); // CAN 1 int vect 4 1 Mbaud
TXBuf.Frame = 0x00080000;     // dlc << 16
TXBuf.MsgID = 0x262;          // id pdor1
TXBuf.DatA  = 0x00050000;     // data 1-4
TXBuf.DatB  = 0x00000000;     // data 5-8
while(!canbase_txmsg(&TXBuf));// invia il pacchetto
```

short **canbase_rxmsg**(CANBASE_MSG *pReceiveBuf);
short **canbase_rxmsg_2**(CANBASE_MSG *pReceiveBuf);

Receives any message from the buffer of the CAN.
The messages on the CAN bus are automatically
received and stored in a circular buffer of 64 messages.
If the buffer is not empty this function Scoda a message from
the buffer and returns withcode 1.
If the buffer is empty, it means that there is no message and
returns with the code zero.

Example:

```
CANBASE_MSG RXBuf;        // Buffer  rx CAN message
U32 can1a,can1b,can2a,can2b;
while(canbase_rxmsg(&RXBuf))
{
   if ((RXBuf.MsgID==((3<<7)|(98))))
   {
      can1a=RXBuf.DatA;  //legge i primi 4 bytes
      can1b=RXBuf.DatB;  //legge i secondi 4 bytes
   }
   if ((RXBuf.MsgID==((5<<7)|(98
   {
      can2a=RXBuf.DatA;  //legge i primi 4 bytes
      can2b=RXBuf.DatB;  //legge i secondi 4 bytes
   }
}
```

CAN-OPEN

The Proteus compiler has a number of functions that allow you to manage thePLC-OPEN CAN stack.

```
void canopen_on(unsigned char sub);
void canopen_off(void);
unsigned char canopen_empty(void);
unsigned char canopen_full(void);
unsigned char canopen_msg(void);
void canopen_flush(void);
unsigned char canopen_rxmsg(void);
unsigned char canopen_txmsg(void);
unsigned char canopen_txmsg_len(unsigned char len);
unsigned char canopen_requestmsg(void);
unsigned char canopen_txsdo(void);
void canopen_onrx(void(*subroutine)());
void buson(void);
void canopen_standard_sub(void);
unsigned char canopen_perif(U8 i,U8 type,void(*subrx1)(),void
(*subrx2)());
U8 send_sync(void);
```

The use of these functions is illustrated in' EX_CANOPEN example.

ETHERNET

The units are based on ARM micro (VK series and S4000)
have an Ethernet interface managed through a set
of functions implemented in the BIOS.
The essential functions are:

General Function (host and server)

void **SetIp**(IP *ip,U16 a,U16 b,U16 c,U16 d);

To implement a Server:

U32 **EMAC_Server**(U16(*subio)(),U32 flags);

To implement a Host

HOST ***CreateHost**(U8 a,U8 b,U8 c,U8 d,U16 port);
U32 **EMAC_Tx**(HOST *host,U8 *txbuffer,U16 flags,U16 txlen);
U32 **EMAC_Rx**(HOST *host,U8 *rxbuffer,U16 flags,U16 rxmaxlen);

To implement a Host or Host/Server

HOST ***CreateHost**(U8 a,U8 b,U8 c,U8 d,U16 port);
U32 **EMAC_Tx**(HOST *host,U8 *txbuffer,U16 flags,U16 txlen);
U32 **EMAC_Server**(U16(*subio)(),U32 flags);

Other function

void **DeleteHost**(HOST *host);
void **emac_perif_on**(int n,int p,int ip,int f,int mo,int ma,int in,int ax,int np);

These functions are sufficient to handle a link and server / client or a local network with other units or VK S4000 and the PC.

void **SetIp**(IP *ip,U16 a,U16 b,U16 c,U16 d);

This feature allows you to specify your IP address.
If a, b, c and d are worth **zero** all command does not change the IP address but **disable the Auto-IP.**
If a, b, c and d are worth all the command does not change the **255** IP address, but still **enables the AUTO-IP.**

The function of **Auto-IP**, enabled by default, allows the device to examine the messages in the network, and automatically enter its IP address as that of a possible **ARP** request, addressed to a node whose IP address is **xxx.xxx.xxx . n**, where **n = 200+ Periph eral**, Peripheral since the number of the device itself (contained in non-volatile RAM).
This mechanism allows nodes to self-present to the local network without the need to set any parameters on the network, but in some cases can be counter productive, so you can disable the program.

The function of **Auto-IP**, enabled by default, allows the device to examine the messages in the network, and automatically enter its IP address as that of a possible **ARP** request, addressed to a node whose IP address is **xxx.xxx.xxx . n**, where **n = 200+ Periph eral**, Perif since the number of the device itself (contained in non-volatile RAM).
This mechanism allows nodes to self-present to the local network without the need to set any parameters on the network, but in some cases can be counterpr oductive, soyou can disable the program.

Example:

```
SetIp(0,0,0,0);          //disabilito auto-ip
SetIp(192,168,1,33);     //setto il mio ip
```

HOST ***CreateHost**(U8 a,U8 b,U8 c,U8 d,U16 port);
void **DeleteHost**(HOST *host);

CreateHost creates a Socket support the
transmission or reception of messages UDP or TCP-IP.

A socket drive contains the IP address of the
recipient, and the TCP or UDP port number.
We need a different socket for each recipient and each port to
which we intend to sendmessages
through the command EMAC_Tx.

The receiving socket may be unique in that we do not
know in advance who will receive messages, or which
port, and then we create it to zero IP address and the port. Will
be receiving the same message to fill in these fields.

DeleteHost used to deallocate the memory
for the host structure.
It can be used to destroy a host of transmission may
be created as a temporary local variable in asubroutine.

Example:

```
Void send_a_message(void)
{
    HOST *h=CreateHost(192,168,1,100,1000);
    char buffer[]="il mio messaggio\n";
    EMAC_Tx(h,buffer,UDP,strlen(buffer));
    DeleteHost(h);
}
```

it sends the message UDP "my message" to
the node 192.168.1.100 on port 1000.

U32 **EMAC_Tx**(HOST *host,U8 *txbuffer,U16 flags,U16 txlen);

This feature allows you to send a message to a node on
a port associated with the host parameter.

Host is the pointer to the socket transmission to use,
identifies IP address and port of destination.
Txbuffer is the pointer to the buffer that contains the message to
forward
Flags can be UDP or TCP, and identifies the protocol to use
for transmission
Txlen is the length of the message to be transmitted

Returns with a value of 1 if the transmission is successful.

The socket Host is used by ARP to associate MAC address to IP
address of the recipient.
If you create a new socket, even if you send a message
to a node that we have already sent other messages with other sockets,
you made a new ARP request to receive the'MAC address of the
recipient, a prerequisite in order to send the message.
Using a global socket, the ARP request will only run the first
time, and the socket will store the MAC address of the destination
node for subsequent messages.
Typically, operating systems like Windows or Linux have their
own ARP stack management, which stores the MAC address of
the nodes for a certain time (usually 5 minutes) after they need it again.
In our case the MAC address,
once acquired, is maintained indefinitely, unless you put
the 6 to 255 fields of the Host address himself to the socket.

Example:

```
Void send_a_message(void)
{
    HOST *h=CreateHost(192,168,1,100,1000);
    char buffer[]="il mio messaggio\n";
    EMAC_Tx(h,buffer,UDP,strlen(buffer));
    DeleteHost(h);
}
```

U32 **EMAC_Rx**(HOST *host,U8 *rxbuffer,U16 flags,U16 rxmaxlen);

This feature allows you to request **a polling** message ethernet.
It is not an essential function, and can
be replaced more effectively by **EMAC_Server** function that does the same thing in an automatic and transparent to the program,without the latter is worrying to consider whether new messages have arrived.

Host	the socket is received. It does not necessarily contain the IP address and port of the message receive
Rxbuffer	is the pointer to the buffer where the function is to copy the received message
Flags	can be UDP or TCP and identifies the protocol
Rxmaxlen	is the maximum length that may truncate the message when you copy it to rx buffer

Return the length of the received message (otherwise 0)

Ethernet is managed by a server, switched to turn on the machine, which handles traffic on the node (ARP, PING, TFTP, etc.)..
When the node receives a message for him, which is not operated by the sever, this is put into a buffer (of 32 messages).
The function EMAC_Rx Scoda any messages in this buffer. We can read in host-> ip,host-> mac and
host-> port the message came from whom and on what port.

Esempio:

```
Void receive_a_message(void)
{
     HOST *h=CreateHost(0,0,0,0,0);
     char buff[100];
     int port;
     if (EMAC_rx(h,buffer,UDP,100))
     {
        port=h->port;
        . . . . .
     }
     DeleteHost(h);
}
```

U32 **EMAC_Server**(U16(*subio)(),U32 flags);

Ethernet is managed by a server, switched to 'turn on the machine, which handlestraffic on the node (ARP, PING, TFTP, etc.)..
When the node receives a message for him, which is not operated by the sever, ifthere are functions of user management posts, it is proposed to the user functions.
If these come back with 0 (no response) the message is placed in a buffer (of 32messages).

EMAC_Server function allows you to define these functions:

U16(*subio)()	is the function of management posts
Flags	UDP (UDP messages according to management) or (TCP message handling function of TCP-IP)

The user function is of type:

U16 function(char *rxmsg,char *txmsg,int flags,int port,int length)

Where:

Rxmsg	is the pointer to the message received
Txmsg	is the pointer to the buffer in which to put the reply message
Flags	can be UDP or TCP
Port	is the gate of the message
Length	is the length of the message received .

It is advisable that the copy function in its own buffer messages and rxmsg txmsg toavoid alignment problems.
If this function fills the buffer txmsg and returns with a value greater than zero, is sent onan ethernet reply message containing the buffer tx msg and a length of the return valueof the function.
If EMAC_Server is used by the client, clearly the user function will handle only handleincoming mail (server response), and return with zero value.

You can only define two functions EMAC_Server time, a type of type TCP and UDP.
At any time we send a new command to replace the fly EMAC_Server reception function with an 'other.

Example server:
On port 1000 if it receives a message
containing a number (in ascii), responds with a
message contenemte twice the number (in ascii)
On port 1001 if it receives a message
containing a number (in ascii), responds with a
message contenemte half the number (in ascii)

```
EMAC_Server(pippo,UDP);

U16 pippo(char *rx,char *tx,int flags,int port,int
len)
{
   int n=0;
   switch(port)
   {
      case 1000:
      sscanf(rx,"%d",&n);
      sprintf(tx,"%d\n",n*2);
      return strlen(tx);
      break;
      case 1001:
      sscanf(rx,"%d",&n);
      sprintf(tx,"%d\n",n/2);
      return strlen(tx);
      break;
      default:
      return 0;
      break;
   }
}
```

The corresponding client could be:
It absolutely the most difficult imaginable to do just the transaction n2 = n1 * 2.

```
int n1
volatile int n2;
char buffer[100];
HOST *h=CreateHost(192,168,1,100,1000);
EMAC_Server(caio,UDP);
 . . . .
```

```
n1=120;
sprintf(buffer,"%d\n",n1);
EMAC_Tx(h,buffer,UDP,strlen(buffer));

U16 caio(char *rx,char *tx,int flags,int port,int
len)
{
   switch(port)
   {
      case 1000:
      sscanf(rx,"%d",&n2);
      return 0;
      break;
   }
}
```

The client becomes the number n1 in an ascii string and sends it to node192.168.1.100 port 1000
The server receives the message, converts the string to a number, multiplies it by two,it converts to a string and sends the reply to the sender.
The client listens to the answer arrival, it converts the string response to a number and puts it in n2, then returns
with zero (no response to avoid an infinite loop).
It is just one example, to multiply a number by two does not do this!

void **emac_perif_on**(int n,int p,int ip,int f,int mo,int ma,int in,int ax,int np);

This is extremely powerful, I can expand a device (or PC) to other devices on the VK series or S4000.
It 's the equivalent of ethernet **can_perif_ext** function of CAN, but much more powerful, flexible and fast.
The parameters:

N virtual slot number. Must be 0 for the first emac_perif_on, one for the next and so on. A emac_perif_on with an equal number in a previous replaces the previous one.
P number of physical device. See Tables 1 and 2
IP fourth byte of 'IP address of the device's physical expansion.
F polling frequency (usually is placed at 1 = 1 millisecond).
Mo excluded form the outputs (32 bits, each bit to 1 does not include the remote control of the corresponding output bit)
Ma excluded form the analog output (4 bits, each bit to 1 exclude the remote control of the corresponding analog output.)
In number of valid inputs for the remote module
Ax axles apply for remote control
Np number of analog inputs are valid for the remote module.

-Tabella 1 (VK)-

Perif(P)	in	out	encoders	pot	anal_out
0	i1-i16	o1-o16	encoder1-2	pot1-4	anal1-2
1	i17-i32	o17-o32	encoder3-4	pot5-8	anal3-4
2	i33-i48	o33-o48	encoder5-6	pot9-12	anal5-6
3	i49-i64	o49-o64	encoder7-8	pot13-16	anal7-8
4	i65-i80	o65-o80	encoder9-10	pot17-20	anal9-10
5	i81-i96	o81-o96	encoder11-12	pot21-24	anal11-12
6	i97-i112	o97-o112	encoder13-14	pot25-28	anal13-14
7	i113-i128	o113-o128	encoder15-16	pot29-32	anal15-16

-Tabella 1 (S4000)-

Perif(P)	in	out	encoders	pot	anal_out
0	i1-i32	o1-o32	encoder1-4	pot1-8	anal1-4
2	i33-i64	o33-o64	encoder5-8	pot9-16	anal5-8
4	i65-i96	o65-o96	encoder9-12	pot17-24	anal9-12
6	i97-i128	o97-o128	encoder13-16	pot25-32	anal13-16

You can mix and VK S4000 devices, provided they do not overlap. For devices VKYou can define multiple encoders (up to 4 each), but no overlap.

It is not necessary that the
devices are contiguous and cover all the 'range of I.
There may be holes, in which case the i /
o for the holes are not valid and should not beused in the program.
The only thing is that the first parameter N in the
first declaration forms starts from zeroand increases without holes.

If you define a module with **P = I 0** the on-board and
remote are disabled on those of the device.

With the command **com_disable (0),** it is possible to
remote device on the first set (the one with N = 0) the local COM.

Limited to the PC, the command **com_disable (0);** on the
first remote device also known as the CAN ports.

In summary, you can expand this function:

- **The digital inputs up to 128**
  The digital outputs up to 128
  The analog inputs up to 32
  The analog outputs of up to 16
  The encoders up to 16
  The encoders up to 16 one-way
  The stepper motors up to 16
  The first form on the COM ports (COM1-COM3)
  The first on the CAN module (CAN1 and CAN2) (* for PC only)

The devices do not require any special software to
operate as slave.E just leave it on the screen, waiting in the
starting connection.

You can also get multiple master devices, provided they do
not operate on the
sameoutput. You can also define devices that are both master and slave
of each other, oncedefined as digital and analog outputs are to be
managed locally and which are remote.

The update times of
the outputs and reading inputs are 1 millisecond, and then to about the
same as the update of the resource / or on board. It makes no
sense for i / ocommand to REMAT __i () and __o (), because
in any case would not be updated immediately.

Example 1: maximum configuration, with **a S4000 compact master and 3 S4000 slave**: Write start.c of master:

```
emac_perif_on(1,2,201,1,0,0,32,4,8);
emac_perif_on(2,4,202,1,0,0,32,4,8);
emac_perif_on(3,6,203,1,0,0,32,4,8);
```

Example 2: maximum configuration, with **a S4000 without I/O and 4 S4000 slave**: Write start.c of master:

```
emac_perif_on(0,0,241,1,0,0,32,4,8);
emac_perif_on(1,2,222,1,0,0,32,4,8);
emac_perif_on(2,4,208,1,0,0,32,4,8);
emac_perif_on(3,6,233,1,0,0,32,4,8);
com_disable(0);
```

Example 3: maximum configuration, with **a PC master and 4 S4000 slave**: Write start.c of master:

```
emac_perif_on(0,0,201,1,0,0,32,4,8);
emac_perif_on(1,2,202,1,0,0,32,4,8);
emac_perif_on(2,4,203,1,0,0,32,4,8);
emac_perif_on(3,6,204,1,0,0,32,4,8);
com_disable(0);
```

Example 4: maximum configuration, with **a PC master and 8 Vk3 slave**: Write start.c of master:

```
emac_perif_on(0,0,201,1,0,0,16,2,4);
emac_perif_on(1,1,202,1,0,0,16,2,4);
emac_perif_on(2,2,203,1,0,0,16,2,4);
emac_perif_on(3,3,204,1,0,0,16,2,4);
emac_perif_on(4,4,205,1,0,0,16,2,4);
emac_perif_on(5,5,206,1,0,0,16,2,4);
emac_perif_on(6,6,207,1,0,0,16,2,4);
emac_perif_on(7,7,208,1,0,0,16,2,4);
com_disable(0);
```

APPENDIX

Comands added to bios 1.7

void **v_copy_line**(int x0,int y0,int dx,int dy,int offset);
void **memcpyfast**(U32 *dst,U32 *src,U32 size);
void **show_arrow**(void *arrow);
void **v_rectfillrotatec**(int xl,int yl,int dxl,int dyl,float a,int xc,int yc,S32 col,int mod);
U32 **fplay**(FILE * file,U32 xs, U32 ys, U32 dx, U32 dy, U32 sx, U32 sy);
void **emac_perif_ext**(int n,int t,int firstmac,int first);
void **emac_perif_on**(int n,int p,int ip,int f,int mo,int ma,int in,int ax,int np);
U32 **emac_perif_init**(void);

Comands added to bios 2.9

int **Step**(U32 n,U32 cmd,S32 op1,S32 op2,S32 op3,S32 op4);

Comands added to bios 2.9b

//**canbase**
short **canbase_init_2**(unsigned short can_isrvect,unsigned int can_btr);
short **canbase_txmsg_2**(CANBASE_MSG *pTransmitBuf);
short **canbase_rxmsg_2**(CANBASE_MSG *pReceiveBuf);
// **can**
void **can_timer_2**(void);
void **can_time_2**(int t);
void **can_perif_2**(int n,int t,int baud);
void **can_perif_ext_2**(int n,int t,int baud,int first);
U8 **can_busoff_2**(void);
U8 **can_msg_2**(void);
void **can_rxmsg_2**(U8 n);
void **buson_2**(void);
void **can_freq_2**(U32 freq);
void **canopen_on_2**(U8 sub);
void **canopen_off_2**(void);
U8 **canopen_empty_2**(void);
U8 **canopen_full_2**(void);
U8 **canopen_msg_2**(void);
void **canopen_flush_2**(void);
U8 **canopen_rxmsg_2**(void);
U8 **canopen_txmsg_2**(void);
U8 **canopen_txmsg_len_2**(U8 len);
U8 **canopen_requestmsg_2**(void);

U8 **canopen_txsdo_2**(void);
U8 **send_sync_2**(void);
void **canopen_standard_sub_2**(void);
void **canopen_standard_poll_2**(int n);
U8 **canopen_perif_2**(U8 i,U8 type,void(*subrx1)(),void(*subrx2)());

void **v_copy_line**(int x0,int y0,int dx,int dy,int offset);

Equivalent to the command v_linec but the fifth parameter, instead of color, specifically the offset in video memory to copy the color pixel.

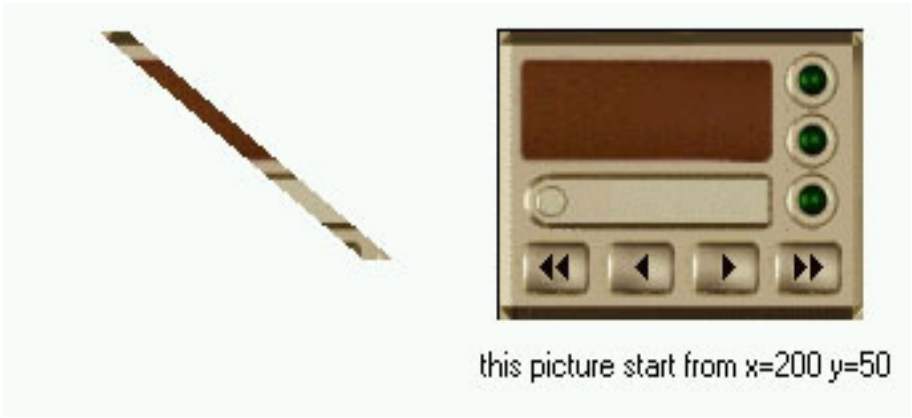
The offset can be calculated as dx + dy * SCREEN_WIDTH.

This command draws a straight line by taking the colors of pixels from an image that is in an other area of video memory.

Example:

```
void hdcopy(void)
{
    int x0;
    for (x0=0;x0<10;x0++)
    {
       v_copy_line(x0+50,50,100,80,150);
    }
}
```

Given the following result:



The pixels of a line running from 50.50 coordinates are taken from the screen coordinates by 150 pixels on the right (200.50)

void **memcpyfast**(U32 *dst,U32 *src,U32 size);

Dst memory address of the destination
Src memory address of the source
Size number of bytes to copy

It is equivalent to memcpy but is faster, making the copy 32-bit (8 bits memcpy copy at a time).

The condition for using this command
is that dst, src and size are multiples of 4.

It is particularly useful to copy parts of the video memory.

Example

```
int vett1[100];
int vett2[100];
 . . . .
memcpyfast(vett2,vett1,sizeof(vett1));
```

void **show_arrow**(void *arrow);

It is a command serves principal for draw hands in instruments.
Refers to a type structure ARROW

```
typedef struct ARROW
{
   int lun;
   int lar;
   int back;
   int col1;
   int col2;
   int x0;
   int y0;
   int dx;
   int dy;
   int traspa;
   float ang;
   float old_ang;
   float min_step;
   OBJ *center_box;
}ARROW;
```

This command can be effectively replaced by the
command v_rectfillrotatec which is much more flexible and easy to use.
For explanations and examples,
refer to 'EXAMPLE_ANIMATION_1 example.

void **v_rectfillrotatec**(int xl,int yl,int dxl,int dyl,float a,int xc,int yc,S32 col,int mod);

It is a very versatile which allows:

- Draw a rectangle rotated around a point
  To rotate an 'image around a point and copying
  an 'other side of the screen
-

Parameters:

Xl	x coordinate of the center of rotation on the form
Yl	y coordinate of the center of rotation on the form
Dxl	width of the rectangle
Dyl	height of the rectangle
A	clockwise rotation angle
Xc	abscissa of the rotation center of the rectangle in relation to 'corner top left
Yc	ordinate of the rotation center of the rectangle in relation to 'corner top left
Color	ram_bitmap or color of the rectangle (depending on the way)
Mod	way. Available in different ways, depending on which changes the meaning of the color parameter:

color way:

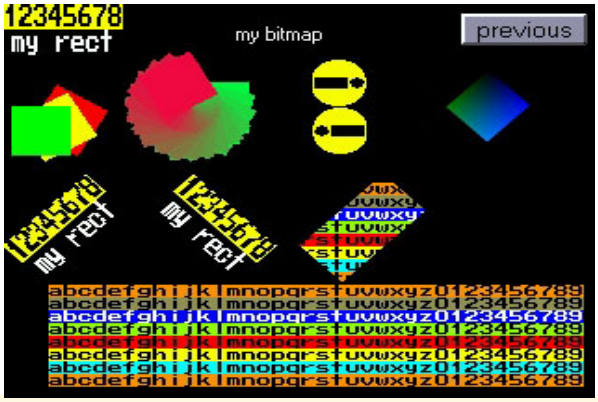
- COLOR	color

rotation way image:

- BITMAP	bitmap_ram address caching bitmaps
- FROMADDR	absolute address of the bitmap in ram
- FROMABSCRT	absolute address of the bitmap in video memory
- FROMABSFRAME	Guideline top screen

image way without rotation:

- FROMRIGIDREL	offset del background in memoria video
- FROMRIGIDABS	indirizzo assoluto del background

The 

functionality of

U32 **fplay**(FILE * fil,U32 xs, U32 ys, U32 dx, U32 dy, U32 sx, U32 sy);

copy video from coordinates xs, ys, reading from
the file filter, dx * dy sx and sy pixelswith scale.
pixels are bytes (CRT_256COLORS) or word (CRT_32KCOLORS)
the coordinates are relative to frame_vect
the file must point to the current frame

This command is not normally used directly, but via the Macro Media:
int **PlayVideo**(char *file,int xs,int ys,int zoom,int sync);
void **PlayAudio**(char *file);
void **PlayVideoStop**(void);
void **PlayAudioStop**(void);

Parameters of PlayVideo:

File name of the media file. If the name has
no extension (ex c: \ \ foo) is searched for thefile pippo.vid the
video and the file pippo.aud for 'any audio. If there is no audio,
just do not load the file on the SD. aud, or specify as a para
meter of the PlayVideo file (c:\ \ pippo.vid).
Xs position of x 'in the top left corner of the movie
Ys y position of the 'upper-left corner of the movie.
Zoom zoom of film (1 o 2)
Sync **sync=0** the movie is run at normal speed
sync<0 shows the frame-sync * totframes/1000. For exam
ple, sync =- 500 shows a frame forhalf the movie.
sync=1 shows the next frame
sync=2...4 reduced speed sync (1 / 2, 1 / 3, 1 / 4)
sync>4 shows the frame number sync.

The function returns the value zero if the file exists, has not reached the end and if theframe set exists, otherwise returns the value 1.

PlayAudio executes a file of type audio.
PlayVideoStop and **PlayAudioStop** stop their functions .

With these functions you
can not see movies like avi or mpeg or play wav or
mp3 audiotype, but must first prepare the audio-video in uncompressed form, using the specialutility

AVICONVERTER.EXE found in the distribution of Proteus in
the c: \ commons \ c \AVIConverter.

If we want to make a sound file, we can convert
it to mp3 or wav, RAW format
11kbps mono 8bit , which is the format used by PlayAudio.

Who wants to see how they are macros PlayVideo
and PlayAudio, the source is located in
c: \ commons \ c \ armgcc \ hitex-arm-elf \ include \ stubs.h

WARNING: PlayAudio use the TimerTask, so it
is incompatible with any programs that use this task.

The examples show EXAMPLE_FILM_VK3 EXAMPLE_FILM_V8 and pr
actical applications of these controls.

void **emac_perif_on**(int n,int p,int ip,int f,int mo,int ma,int in,int ax,int np);
void **emac_perif_ext**(int n,int t,int firstmac,int first);
U32 **emac_perif_init**(void);

Emac_perif_on is illustrated in the chapter on the Ethernet
Emac_perif_ext and **emac_perif_init** are present only
for compatibility and are not to be used in new projects.

int **Step**(U32 n,U32 cmd,S32 op1,S32 op2,S32 op3,S32 op4);

This command allows you to manage the movement of stepper motors, ensuring theproper acceleration and deceleration,
the correct speed and handling for each motoroutputs ENABLE, DIRECTION and STEP.
The maximum operating frequencies are 80KHz in case of a motor unit,
40 KHz in the case of 2 engines, 20 KHz for 4 motors.

Parameters:

N engine number. The VK series handles up to two engines for each unit, while the S4000 series will handle up to 4. If you use the 'Ethernet expansion (see the command emac_perif_on) it is possiblefor the' master management unit directly to the stepper motors of the slave, up to 16 engines. for this N can take values between 1 and 16
Cmd You give the command to the motor (see table below)
Op1 first operand (see table below)
Op2 second operand (see table below)
Op3 third operand (see table below)
Op4 fourth operand (see table below)

Each engine has a support structure like this:

```
typedef struct STEP_MOTOR
{
    S32 quota;    //quota attuale
    S32 target;   //quota da raggiungere
    S32 target1;  //quota fine acc.
    S32 target2;  //quota inizio dec.
    U32 rampa;    //rampa acc. e dec. in step
    U32 speed;    //v. max in (clock*8 @vmax)
    U32 status;   //0=non ist. 1=fermo 2=acc. 3=regime 4=dec. 5=stop
    U32 passo;    //passo parziale rampa
    U32 t;        //tempo attuale semiimpulso (in usec/8)
    U32 k;        //livello digitale UP-DOWN
    U32 overrun;  //vel. magg. di calcolo
    S32 ud;       //up/down
    U8  *onoff;   //usc. di onoff
    U8  *dir;     //usc. di dir
    U32 oldrampa;
    U32 oldspeed;
    U32 *tab;
}STEP_MOTOR;
```

Command table for the motor:

Comando	op1	op2	op3	op4	return
1=init	onoff	dir	quota	–	&quota (0 se error)
2=go	quota	speed	rampa	wait	1 (0 se posiz in corso)
3=delta	delta q	speed	rampa	wait	1 (0 se posiz in corso)
4=stop	–	–	–	–	1 se pos. in corso, else 0
5=busy	–	–	–	–	1 se pos in corso, else 0
6=errors	–	–	–	–	errors
7=query	–	–	–	–	&step_motor
8=0ff	wait	–	–	–	–

In the command **init** denote by **onoff** and **dir** the number of corresponding output (eg if onoff =o1 we put onoff = 1)

Speed is given in pulses per second.
Ramp is the ramp (number of steps to speed) acceleration and deceleration, and is indicated in pulses (0 = no ramp, maximum = 5000)
Wait is the wait time in milliseconds between **Enable** and **Dir**
& step_motor is the pointer to the support structure of the engine.

The outputs of the clock are the same as analog outputs (you must select a jumper on the card to switch the function)

vk3-v5-v8-v10:

Step1	PWM0_4	P3.19
Step 2	PWM0_5	P3.20

s4000

Step 1	PWM0_1	P1.2
Step 2	PWM0_2	P1.3
Step 3	PWM0_3	P1.5
Step 4	PWM0_4	P1.6

For more information please see the example EXAMPLE_STEP_MOTORS

short **canbase_init_2**(unsigned short can_isrvect,unsigned int can_btr);
short **canbase_txmsg_2**(CANBASE_MSG *pTransmitBuf);
short **canbase_rxmsg_2**(CANBASE_MSG *pReceiveBuf);
void **can_timer_2**(void);
void **can_time_2**(int t);
void **can_perif_2**(int n,int t,int baud);
void **can_perif_ext_2**(int n,int t,int baud,int first);
U8 **can_busoff_2**(void);
U8 **can_msg_2**(void);
void **can_rxmsg_2**(U8 n);
void **buson_2**(void);
void **can_freq_2**(U32 freq);
void **canopen_on_2**(U8 sub);
void **canopen_off_2**(void);
U8 **canopen_empty_2**(void);
U8 **canopen_full_2**(void);
U8 **canopen_msg_2**(void);
void **canopen_flush_2**(void);
U8 **canopen_rxmsg_2**(void);
U8 **canopen_txmsg_2**(void);
U8 **canopen_txmsg_len_2**(U8 len);
U8 **canopen_requestmsg_2**(void);
U8 **canopen_txsdo_2**(void);
U8 **send_sync_2**(void);
void **canopen_standard_sub_2**(void);
void **canopen_standard_poll_2**(int n);
U8 **canopen_perif_2**(U8 i,U8 type,void(*subrx1)(),void(*subrx2)());

These commands manage the second CAN interface of S4000 series and are identical to commands without the suffix _2, the only difference being that the second command line instead of the first CAN.

The two CAN lines are identical and completely independent.