Visual I/O

Productivity tools

Programmer's Guide

- **Editor**
- **Experts**
- **Language**
- **Advanced programming**

Ver 7.xx

Philosophy of Visual I/O

Visual I/O is a development tool for generating Man/Machine interfaces applications running on a PC compatible computer.
Visual I/O integrates a real-time engine allowing executing programs written in Grafcet, ladder (with Visual PLC) and high-level language. Programs written in Grafcet and Ladder by Visual PLC are the companion tool for automation applications.

Visual I/O can connect PLCs thanks to communication specifics drivers or by its OPC client interface (see OPC Settings in main menu of Visual I/O).

To insure a permanent dialog, these programs are executed in back task. You can also use threads.

Visual I/O decomposes into two big parts :

A parameter part, placing graphics objects (Circle, Buttons, trends etc.) set up them through dialog boxes.

A second part is pure programming. In this part you can program specific actions, these action can't be predefined in simple dialog boxes.

Visual I/O integrates the ARSOFT's compiler. This one is extremely fast, generated application entirely compiled.
Visual I/O does not need any other tools (like VBA) for the development as well as execution of your applications;
Visual I/O generates executables files (.EXE) including all the files, which simplifies all the installations and reinstallations.

The programming language chosen is the Pascal language. This one allies power and simplicity. About 1000 intuitive functions are available in programming mode. This language will be used to create high-level symbols, which you will place in your editor toolbox.

Visual I/O allows the constitution of an application including one or several screens called Forms or mimics. Every form will contain graphic elements (Circles, Buttons, Curves).

Expert mode:
By double clicking on every graphic element, a dialog box appears. For every element, Visual I/O lists its possible features (ex: a button can be simple, toggle, associated to a key etc.).

Program mode:
For cases which cannot be foreseen in expert mode, the programmer can use the programming mode by selection of the graphic element then a click right with the mouse and choosing programs item in the floating menu. A program editor with syntaxes colouring enables the user to write program code associated to the graphic object.
This code is compiled automatically 'on the fly ' by the ARSOFT compiler integrated into Visual I/O.
This compiler is optimized and as successful as the most known compilers.

Drivers for PLCs:
Due to the various experts, you will be able to define dialogues under various protocols (Modbus, TCP/IP, SysmacWay, MPI Siemens Etc).
These drivers use variables defined in the Visual I/O editor.

These drivers are executed only in the real-time engine (in windows back task), allowing a communication with a maximum possible fluidity whatever the treatments run in front task (at user level). For other task you can also use the "Threads library".
The final project is compiled so the application you develop is deliverable as an executable file (EXE), including all the necessary files. The additional files can be added manually in this executable (Menu *utilities/Create an executable*). When you launch your application from the editor, this one compiles it. If the key (dongle USB) is plugged on the PC, the executable created is free of charge at runtime and duplicate as it. If the USB key is not present at the PC, the executable created is in demonstration mode. This demonstration mode displays a dialog box which announces the mode at launching time, then stops at the end of 15 minutes.

I have no time to read this book

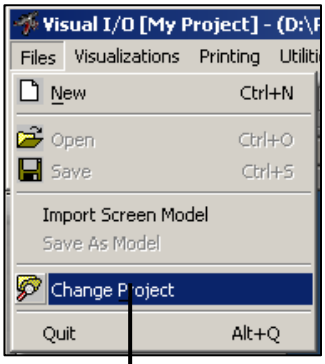
1 - Creating a new project

Under Visual I/O, projects are indicated by names and are stored in the Windows © registry as well as the directories associated containing all of your application files.
No other directory will be used any more for saving files. For the complete safeguard of your application, it is easy for you to copy all the contents of the working directory.

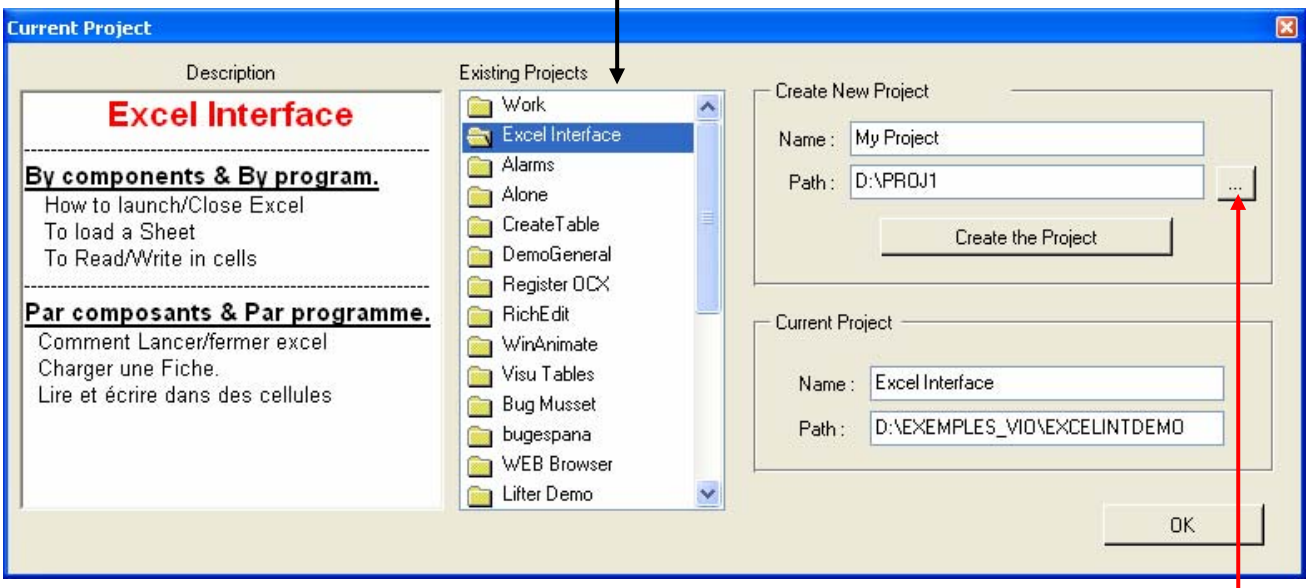
2 - Defining the name of the project

In the Files menu, click on Change Project for

- Changing Project
- Creating a new project



Double click on the project name to set it as the current project



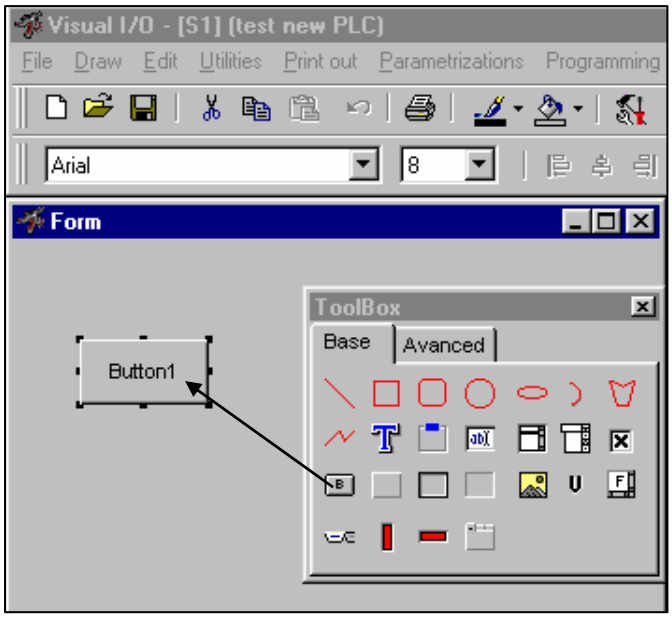
Indicate in the field *Name*, the name of your new project (here My Project)
Indicate in the field *Path*, the complete path where is located in this new project (here D:\Proj1).
If the directory already exists or misses, you can navigate, choose or create it while clicking on this button

Then click on the **Create the Project** to record the name and path for this one in the Windows registry.
Then this project is listed in the central list box, it is easy to select it by double-clicking on.

☞*Note:* When you create a new project, no global variable is defined. You must define them.

The description of the project (optional) is possible on the left editbox in RTF format.
To obtain different graphic Font and graphics effects, constitute your text under Word then Copy/paste it into the Description zone.

3 - Placing a button on the screen

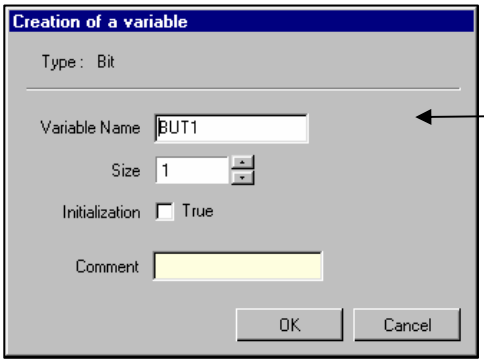


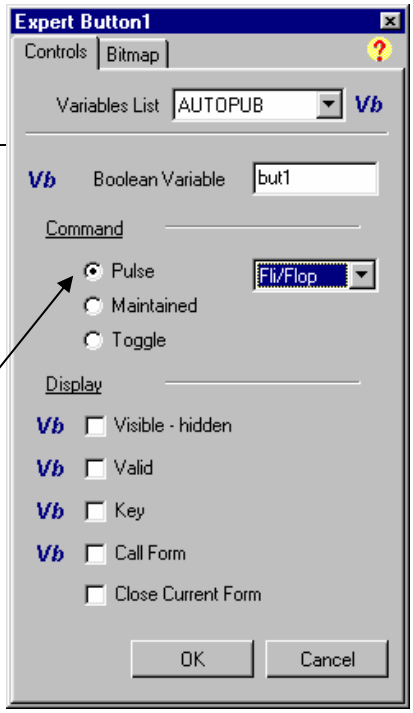
Call the toolbox, choose a button then place it on the screen.

Double click on this button to show its dialog box for setting the parameters.

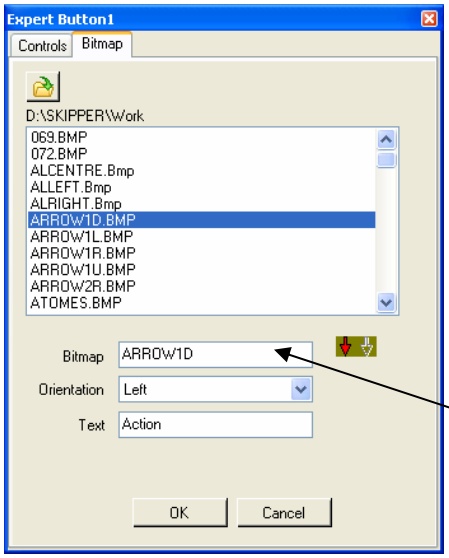
Enter the variable you want to affect to this button. (Here **But1**)

Here the variable **But1** is not defined. The system creates it automatically through a dialog box which enables to determine its size.
(1=Variable, > 1 =Array).
An initialization is possible when the Initialization checkbox is checked.





Here the button is a Flip/Flop, the state of **But1** will be inverted by each mouse click.



Visible - hidden: Bit according to its state to show or hide the button.
Valid: Bit to enable or disable the button.
Key: Associated key to the button.
Call form: Form name called by the button (option).
Close current Form: The button can close the current form.

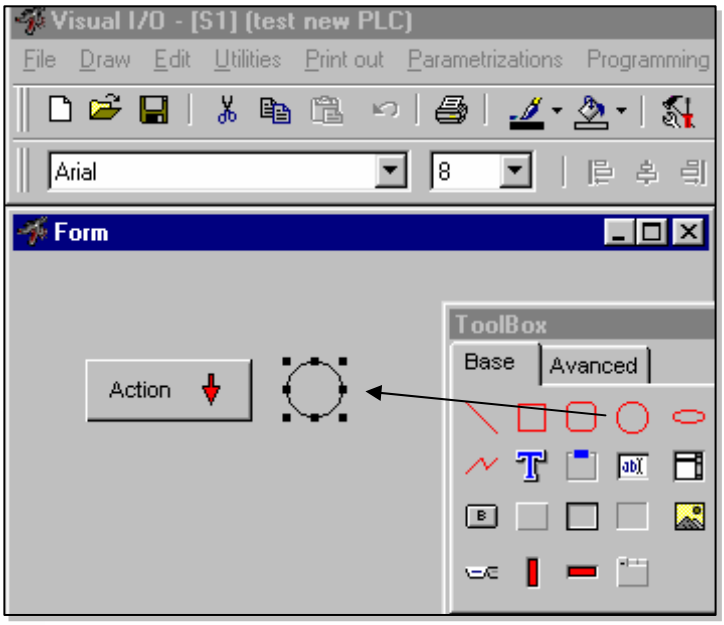
Finally set the button caption.
There is a possibility to display a small Bitmap on the button to represent the action.

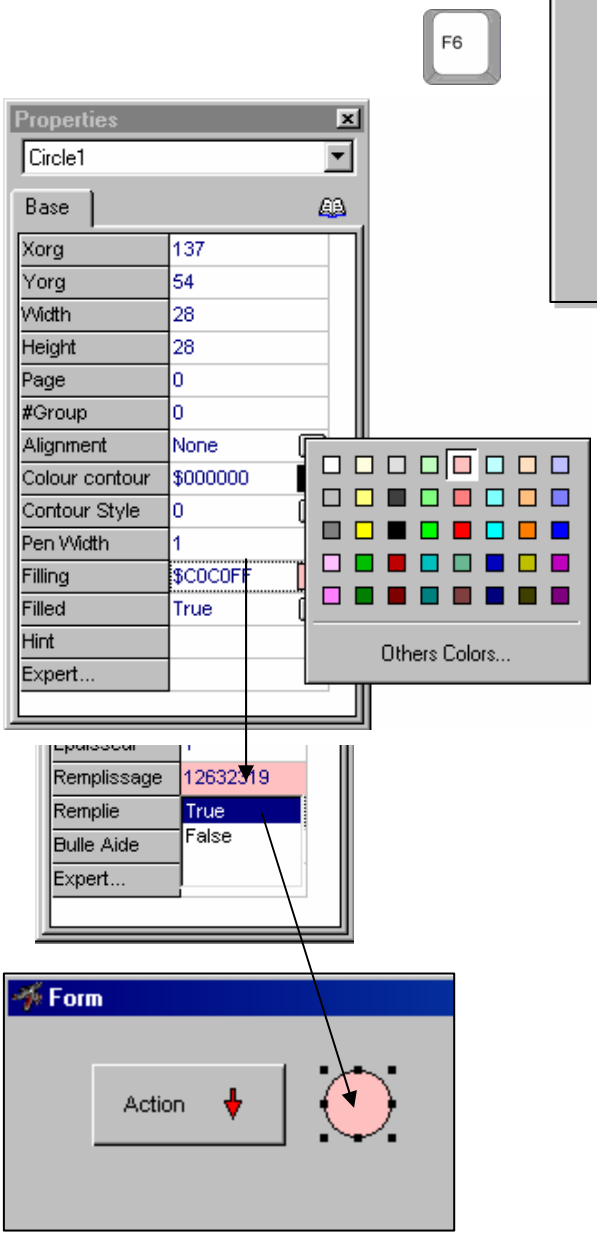
4 - To display the state of a bit (Boolean) by a circle

Select a circle, a rectangle, an ellipse or polygon to display the state (but1) affected by the button.

The circle, rectangle, ellipse or the polygon have the same graphics properties.

However, these shapes are not filled.





To choose the default color (without animations), select the circle (or other shape), then press the **< F6 > key** to show the **graphic properties**.

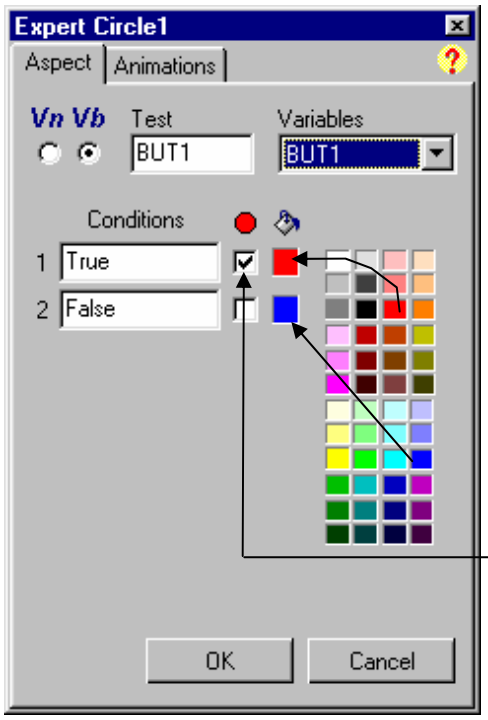
This properties box is containing the various adjustable graphic attributes of the selected component.
These attributes can be various, according to the component type.

To choose the default color for this component, double click on the cell in front of the text *<filled>*.

To fill the component or not, double click on the cell in front of the text “filled”.

5 - Setting the circle parameters

Double click on it, then a dialog box for setting parameters is shown.



Sent by Drag and Drop, the variable **But1** from '*Variable'* Combo Box to the field '*Test'* strike directly the name of the variable to test.

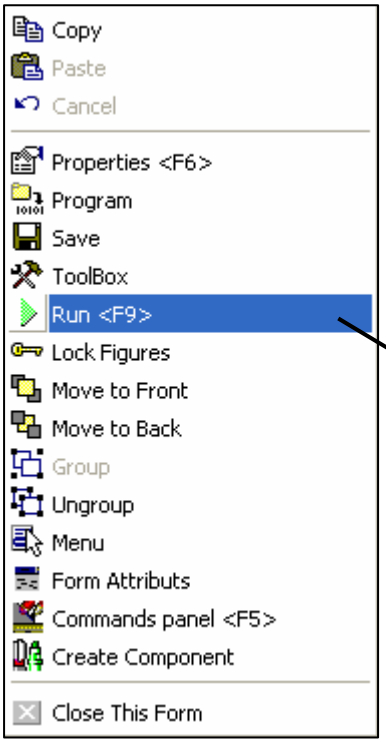
Drag and Drop : Click with the left button of the mouse the text **But1** in Combo box , keep the button on, move the mouse cursor over *test* field then release the mouse button.

For setting colors, bring the chosen color by drag and drop from the palette to the square.

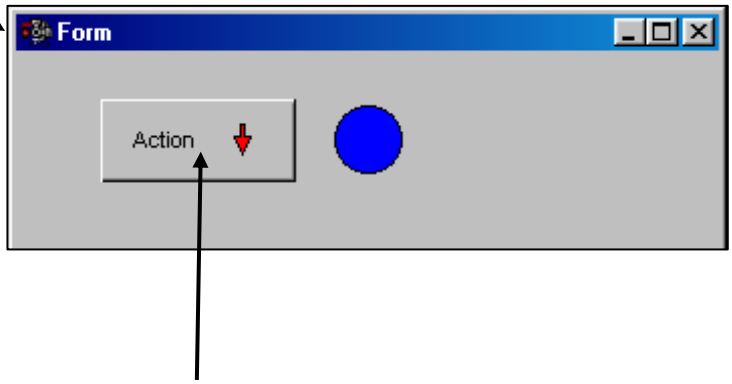
A color is affected not only when the bit is true but also when this bit is false.

The checkbox allows to blink the component when the condition is valid (checkbox state checked)

6 - Go in Run mode

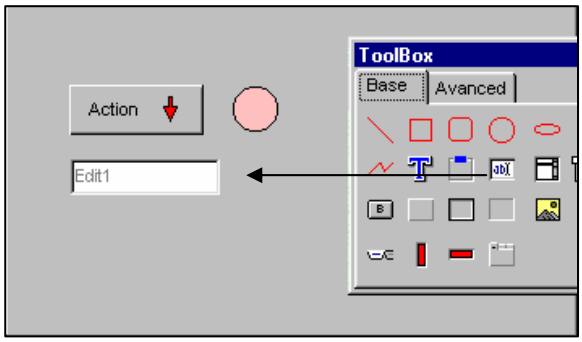


*To launch your first application containing a single window, click with the **right button** of the mouse then select **Run** **or press <F9> key.***
Run also compiles this application and create an executable file.

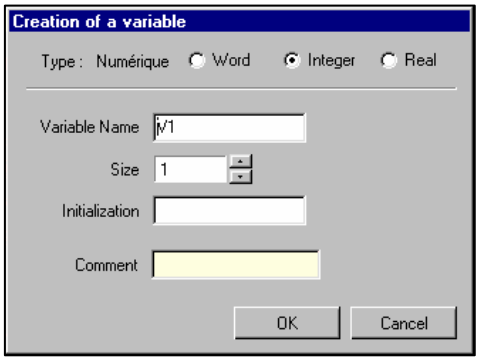


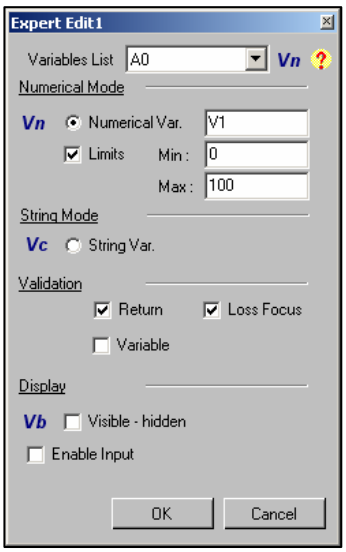
Click the button, variable **But1** is set and animates the circle.
A second click of the mouse affects **But1** to False and animates the circle with another color.

7 - Creating an Edit box



Select the component Edit box in the toolbox. Then place it on the screen.
Double click on it. An expert is shown.





Click the radio button <u>Numerical Var</u>., to enable editing at run time to be a numerical value.
Here we want to enter a value in variable V1. However, this variable does not exist.
A dialog box allows the declaration of this variable.

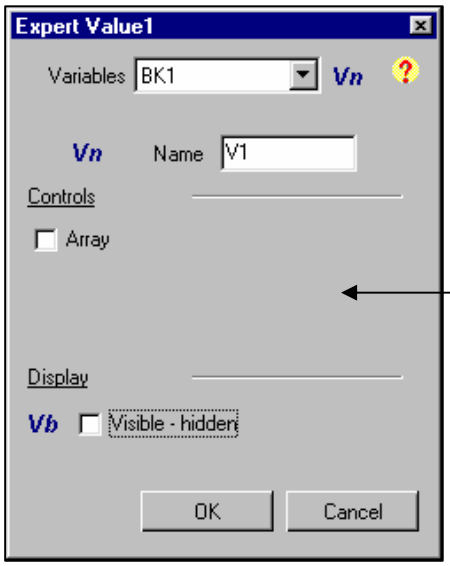
Three Types are possible **Word=16 bits**, **Integer=32 Bits or Real=Float**.

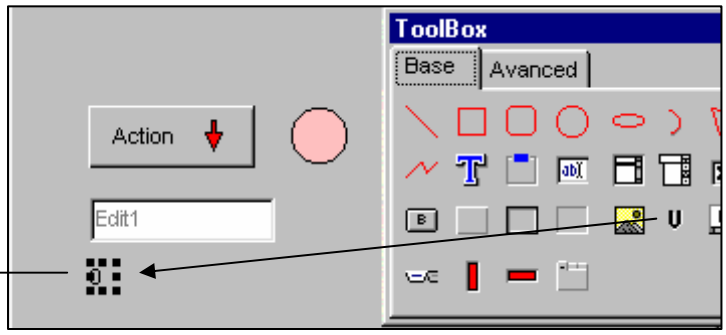
As all the other variables, an initialization is possible and this variable can also be an array. This variable in editing can be limited, then click the 'Limits' checkbox, set the Min and Max parameters (Here 0 to 100 maxi.)

Choose the type of settings for the variable V1. On **Return,** means that the value entered in the field will affect V1, on pressing the <Entry> key or on loss focus from the field (for example, jump to another edit box).

8 - Display a numerical value

Select the 'Numerical Value' component in the toolbox, and then place it on the screen





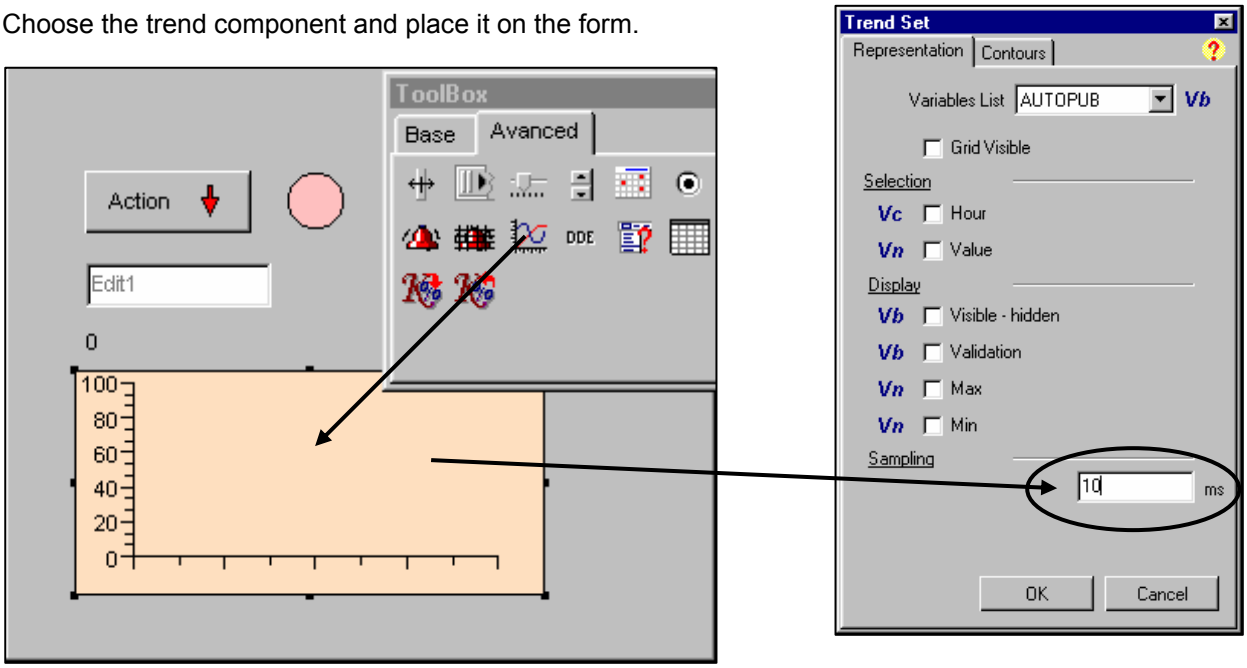
A dialog box appears, allowing setting the variable to be shown.

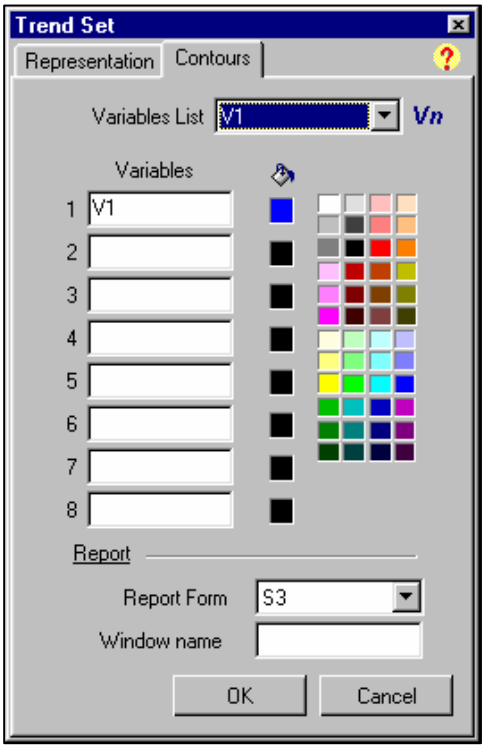
Here choose variable **V1** to show.
This variable is affected in the field Edit1 that set previously.

On every graphic component, a checkbox is present to realize Visible Hidden attribute for this component, according to the state of a Boolean variable.

9 - Display Trends

Choose the trend component and place it on the form.





Choose the sampling period, Here 10 ms, then go under the *Contours* tab to allocate the first pen on variable **V1**.
Each trend contains not more than eight Pens (eight Curves).
Two checkboxes also allow affecting numerical values for Maxi and Mini on the X-axis.

Each curve has its color. Also,you can call others properties by pressing **<F6>** Key.

Note : Another possibility is to use the trend-viewer.
See chapter the Real-time data dynamic plots in this book.

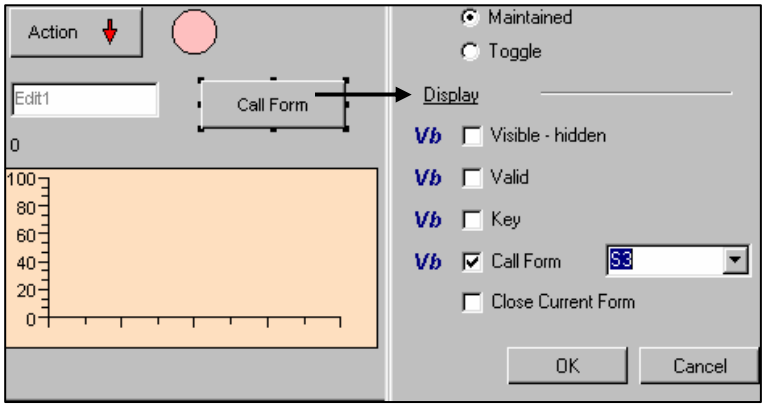
10 - Creating an application with several forms.

Visual I/O enables applications to contain several (forms) screens. In any application there are a main form and secondary forms. The main form is the form that will be show at the launching time of your application. There are several methods to call secondary forms from the main form.

1. Call by button
2. Call By Menu
3. Call By program.

Call by button:

Place a button on the screen, and then double click it. One dialog box appears

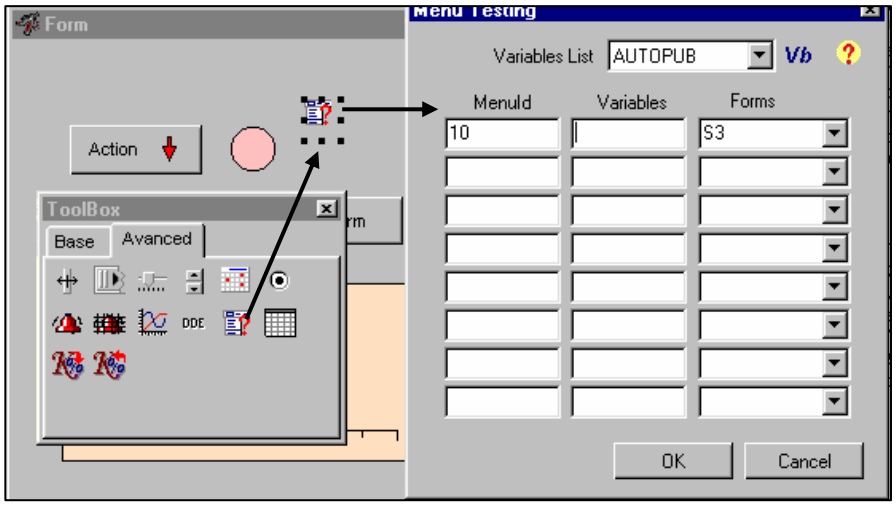


Set the checkbox *call form*, and then choose the name of the form the component must show.

If the form is not created yet, enter its future name.

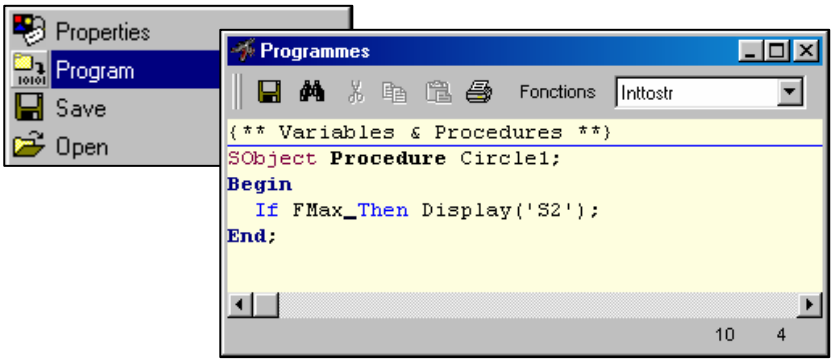
By placing a name in the field Call Form, the system adds automatically this form in the final application list.

Call by menu



Place the component **Test Menu** on the form then double click this one.
A box allows setting a menu identifier to call a form.
In this component, it is also possible to set a Boolean variable.

Call by program



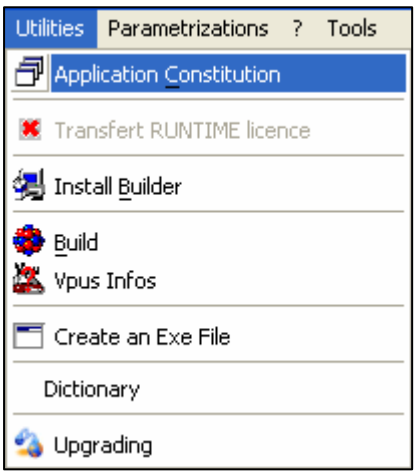
Select a component, and click with the right button of the mouse.
In the popup menu, click on *Program*.

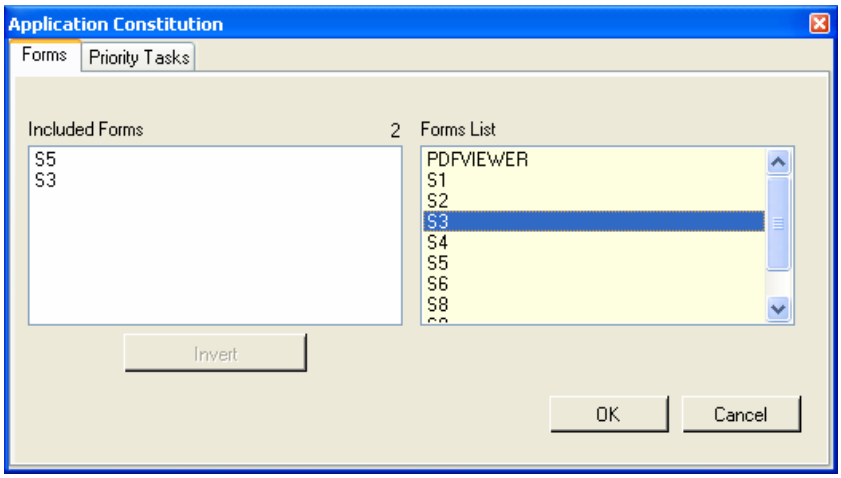
A code editor is displayed.
Enter **If** FMax **Then Display** (' S2 ').
If the Boolean variable (FMax) is true, then the system shows the form named S2.
See also DisplayModal instruction.

11 - Addition and modifications of the list of forms in your application

You can add, erase a form name in the final application by calling 'Utilities' and Constitution of application'.

A window is displayed:





The list of the forms name included in the final application is shown in the left list.
To erase, choose a name then press the <***Del***> key or **double click**.

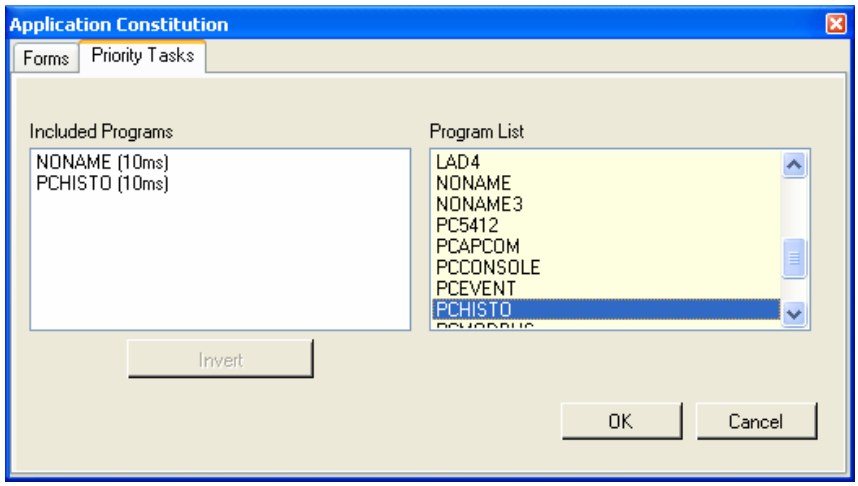
To add a form in the final application, bring by dragging and dropping the name of the form chosen from the right list to the left list box or **double click on its name**.

Under *Priority Tasks* folder, Principle remains the same.

.

If you have Visual PLC, you can create programs in ladder and Grafcet languages, and then carry them out in the Real time engine.

The time base of the programs is indicated between brackets. You can modify this time base in utilities/Vpus info.



Two files are created:

RUNVIO.PFC for the list of the forms (Forms Folder).
PROG.OCO for the list of the programs running in the real time engine of visual I/O (Priority Tasks folder).

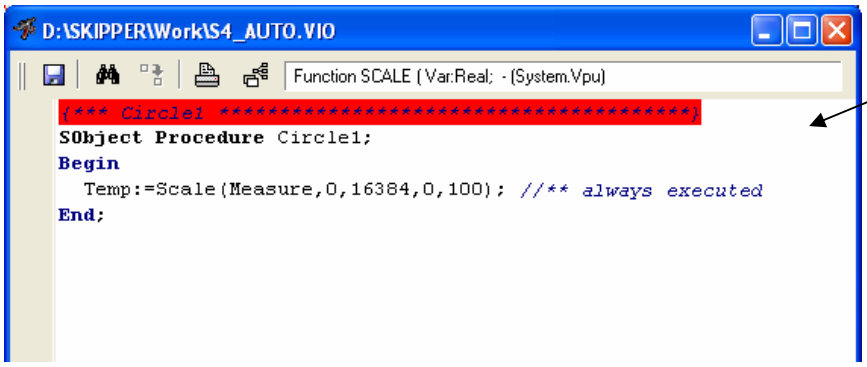
12 - Add a little program in my application.

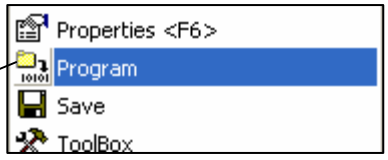
To add a program into your application there are 3 ways:

- The program is executed only when a specific form is displayed on the screen.
- The program is always executed on any form.
- The program is always executed in background in your application.

Example: Scale conversion Analog input (0 to 16384 in 0 – 100 °C)

The program is executed only when a specific form is display on the screen
Choose the form containing the program; select a component on the form, for example, a circle, button edit box, any component, and then click right with the mouse on *Program* item.

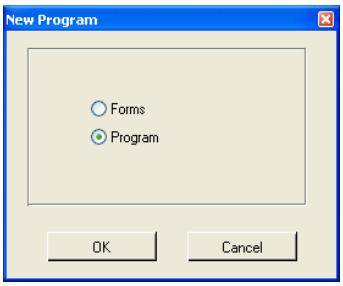




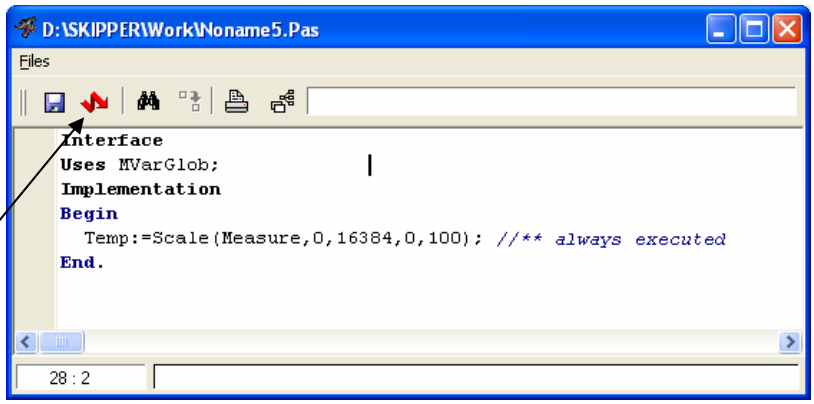
Here the **scale** function is executed periodically each time the form S4 is displayed. This program does not refer particularly to the circle 1. The circle is only used as a program container.

The program is always executed on any form displayed.
You must choose the main form. The main form is the first form displayed at the launching time of your application. This form is listed on pole position in the list of constitution of application (see in main menu). Concerning the rest, it is exactly the same thing you have seen before.

The program is always executed in background of your application.
You must create a new file containing this program (in our example below it is only one line). Click on New button then choose Program. A name of program is automatically generated and a text editor is displayed.



After writing your program, you must compile it by clicking the button with the red Zigzag.

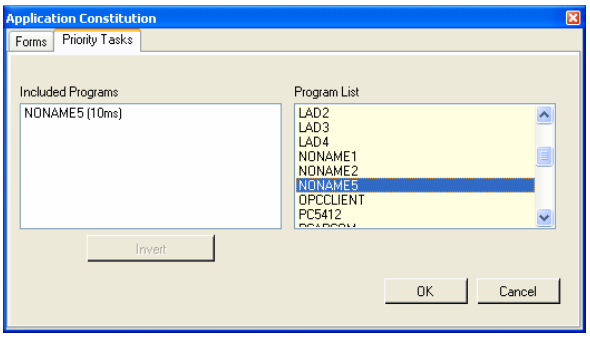


Normally the editor is displayed with **written Unit** in the first line. You must clear this line to transform it in a cyclic program. The simplest expression of this program is to write your instructions between **Begin** and **End** called *the body of this program*. This body will be periodically called by the system (10ms).
After compiling, you must add this program in the list (Here the program name is Noname5).

Add to the list of your application programs.
The compiled programs are listed in the right list.
To add a new program before complete compiling
Bring it by drag & drop or double-click on its name.

For details, see the chapters

- Programming
- Creation of VPU libraries or cyclic programs



Variables Types

Under Visual I/O, you can use different types (system) of variables:

Numerical types

Number without point

Integer	: -2147483648 to 2147483647	(32Bits 4 bytes)	*Ex : I:=123456;*
Word	: 0 à 65535	(16Bits 2 bytes)	*Ex : W:=60000;*
Byte	: 0 à 255	(8Bits 1 byte)	*Ex : B:=128;*

Number with point

Real	: 1, $9x10^{-4951}$... $1,1x10^{4932}$	(10 bytes)	*Ex : R:=243.45;*
Single	: 1.5 x 10e–45 ... 3.4 X 10e38	(4 bytes)	*Ex : S :=-234.56 ;*
Double	: 5.0 x 10^–324 .. 1.7 x 10^308	(8 octets)	*Ex : D:=345.56;*

String type

String : string of 255 characters. (From 1 to 255 characters) 256 bytes. *Ex : S :='ARSOFT' ;*

Boolean type

Boolean : Octet (True=1 False=0). *Ex : B:=True;*

Visual I/O allows addressing Bit in a word. *Ex : W10.0 :=False (First bit of W10).*

Array type

Array : array of type defined before. *Ex : Array [0..12] of Boolean;*

Defining new types

You can redefine the name of a basic type or creating a new type.

```
Type    //*** Type is a language reserved word *****
  Handle : Integer ;
  Bool   : Boolean ;

//*** Program's Var Section or library allow the declaration of variables
Var
  H  : Handle;  //*** equal to H : Integer;
  WB : Bool;
```

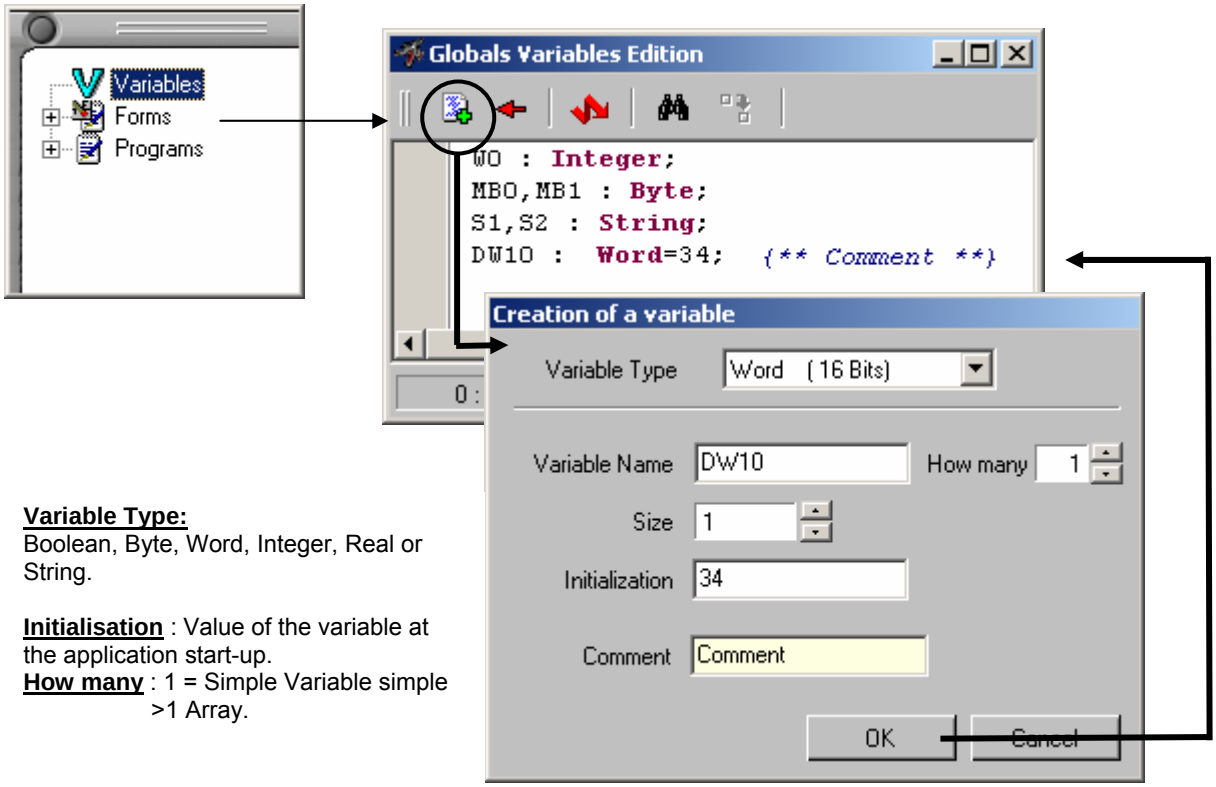
Record type

```
Type
TPoint = Record
          X : Integer;
          Y : Integer;
         End;
Var
 Point : TPoint;

//*** using var point *****
Begin
  Point.X:=12;
  Point.Y:=56;
End;
```

Creating global variables

Call the global variables editor

In the navigator, double-click on *Variables*. A text editor appears.
Then you can directly write the variables names and their types or use the expert allowing adding the variables
Here DW10 is adding variable at the end of the current variables declaration.



Variable Type:
Boolean, Byte, Word, Integer, Real or String.

Initialisation : Value of the variable at the application start-up.
How many : 1 = Simple Variable simple
>1 Array.

Compiling the Global Variables

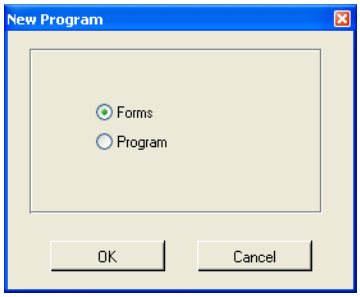
Files created

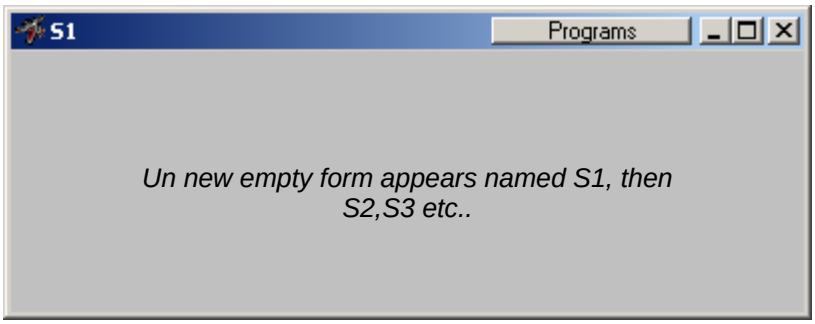
Source File : Declaration of the variables MVARGLOB.PAS
Final File : After compilation of the source file mvarglob.pas, the file MVARGLOB.VPU is created.

After each compiling, the status bar informs you of the compiling result if this compilation is OK.

Creating a new form

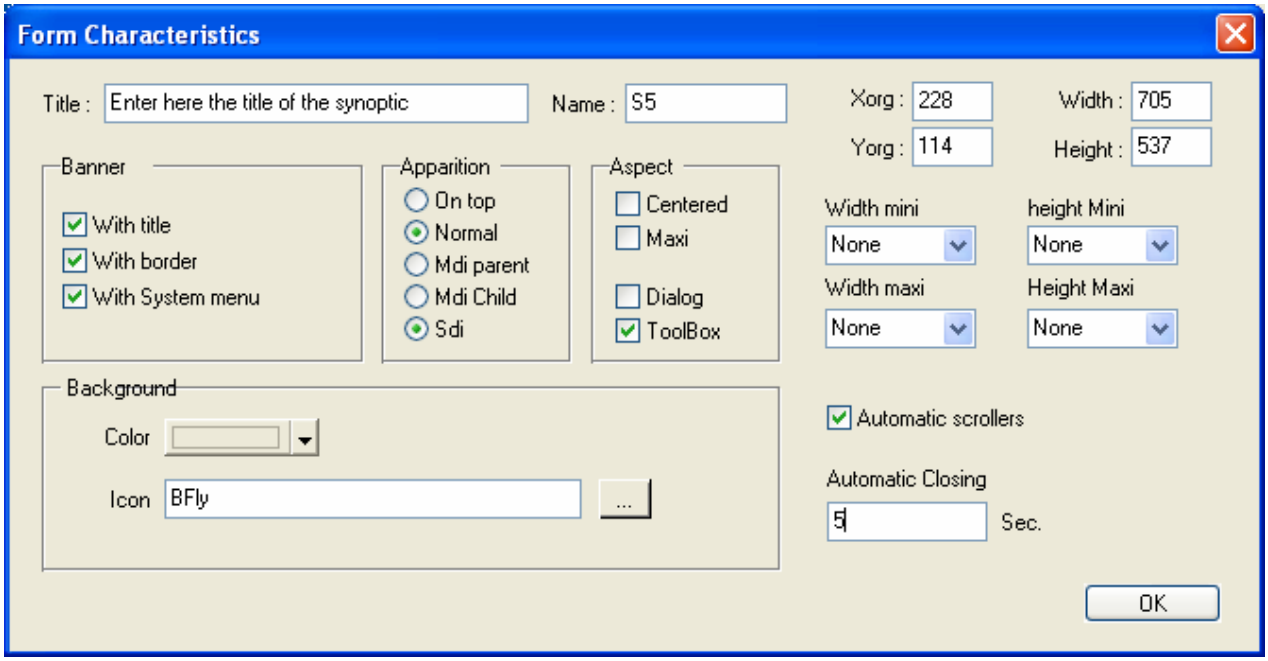
In the menu *Files*, click on *New* then choose Forms then click the OK button





Form attributes.

Double click on the form background or in the popup menu click on *Form Attributes*



In the **Title** field, enter the text appearing in the title bar of the form (caption).
Dialog and toolbox are functionalities concerning the programs in the form.
By checking one of these options, programs included in this form will be executed if the window has the focus (Title bar active).
Mdi (Multiple Document Interface) are those windows reside under a single parent window, while **Sdi** is a single document interface and a method which organizes graphical user interface applications into individual windows. So the operating system's window manager can handle the windows separately.
In Mdi mode, one window, usually called the MDI parent or MDI container, contains many other windows, usually called child forms. SDI mode is generally used.

Title	Defined the title of the window's title bar.
Name	File Name of the form (extension .SYN).
With Title	Allows displaying the window with or without title.
With border	Allows displaying the window with or without allowing border resizing.
Menu System	Defined a system menu in this window or not. The system menu is the small buttons at the top or right of the title bar.
Xorg,Yorg	Absolute Coordinates from the top left corner of the window.
Width, Height	Width and Height of the current form.
Background	Palette of Colors to set the background color. To choose a non defined color, double click on this one then a palette of colors will appear.
Icon	Specifies the icon that appears when the form is minimized.

Width Mini — to define the possible **Minimum** Width when the user resizes the window.
The different options are valued, None or default.

Height Mini — to define the possible **Minimum** Height when the user resizes the window.
The different options are valued, None or default.

Width Maxi — to define the possible **Maximum** Width when the user resizes the window.
The different options are valued, None or default.

Height Maxi — to define the possible **Maximum** Height when the user resizes the window.
The different options are valued, None or default.
Numerical Value : in pixel (>0)
None : No restriction size.
Default : The current value (Value of width field).

Automatic scrollers — Allow displaying horizontal and vertical scrollers in case of resizing of the form.
If this option is checked, no scrollers will be displayed.

Automatic Closing — Allow closing the form automatically if no events (keyboard or mouse) is performed during the temporization defined.

Windows types

Appearance**:**

On Top — This form remains on top of the desktop and of other forms in the project, except any other form that also have FormStyle set to OnTop. If one OnTop form launches another, neither forms will consistently remain on top. This is the case of the toolbox of Visual I/O.

Normal — Allows the form to be covered by another one. It is the inverse on Top option.

MdiParent — Visual I/O applications can be MDI or SDI applications.
An MDI application needs only one Parent window and multiple Child windows.
If this option is Check, indicate that your form is the main form and it is the Parent window of an MDI application. If several parents are set in an application, the first will be exactly the parent. The others will be attributed children by the system.

MdiChild — Defined your form as a child of an MDI application.
If this one contains a main menu, this one will replace the similar menu during its display.

Sdi — The form is not a part of an MDI application.
It has no Parent Window. A SDI form can be included in an MDI application. In this case the form will not respect MDI rules. The form can go everywhere in the screen.

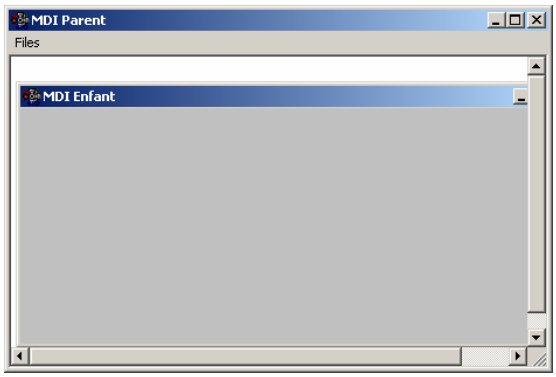
Centred — This option allows the form to be centered in the screen at its first display.

Maxi — Enlarge the form at the maximum size as large as possible (the entire screen).

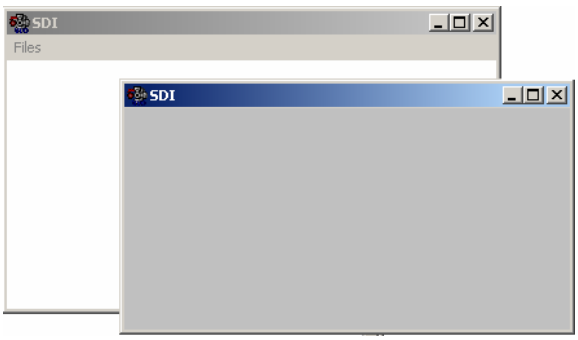
Dialog — Defined the form as a Dialog Box. It allows showing a different presentation from the window.
It allows executing internal programs in this form only when the window has the Focus (title bar active). Concerning execution optimization (speed) of the application, to set forms in Dialog attribute allows sharing time.
Certain programs of form do not need to be executed if the window is not active it does not has the focus.

ToolBox — This option allows showing a presentation of the form as a toolbox.
The main bar is reduced and only the close command of the Window appears.
This option has the same characteristics as the Dialog option.

.



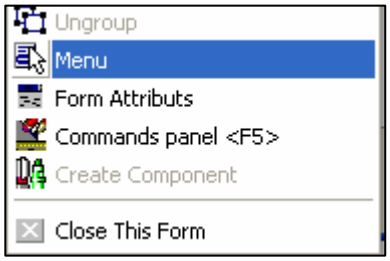
MDI Application, *the main window is a container for child window.*



SDI Application, *the main window and the others windows of the application are free to move across the screen.*

Generate a main menu

It is possible to add a main menu to your form.
If you create SDI applications (Simple Documents Interface), different forms have their own menu, those will be always attached to their respective form.
In MDI applications (Multiples Documents Interfaces), the main form which contains a menu and menus of the children form (if existing) comes to replace the main menu during their displays.
Creating a main menu is called *Menu* in the floating menu



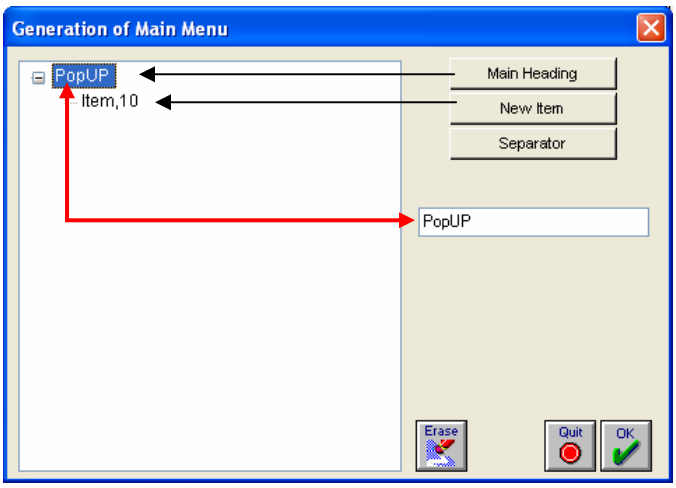
A dialog box appears with an empty List Box on the left.
This List Box is a tree structure symbolizing the various elements of the main menu.

Each main heading can contain elements called Items.
In these main headings we will be able to insert others (sub-menus) containing also items and so on.

Items are labels, which will appear under the relative heading. In addition to the items, we can insert separators (horizontal line without text). Separator divides the items in a menu into logical groups.
The items will have **id numbers** allowing tests by program. Some instructions such as **SetMenuString** or **ToggleMenu** modify (by program) the aspect of these items at Run time.
During the creation, a new item (single number) is assigned to it. However, this number must be unique in order to be able to test in Run. The test is performed by the component such as **TestMenu**.

Heading titles do not have any identifiers. In Run to address them (**SetPopupMenu** procedure) we have the following convention from the left the heading carries number 0. Its neighbour of right-hand side carries number 1 etc.
The headings or the items can have letters for call (shortcut). Menu shortcuts that appear on menus give the user an alternative way to select menu commands using such as the keyboard. For example the file heading has the letter F for call.
To determine a shortcut you can place the character **&** before the character.
Place the character & before the letter chosen at for effect to underline the letter and to be able to directly call in Run.
The shortcut is a combination of the **< ALT > key + < the selected letter >.**
For example, call the heading file without clicking with the mouse, only press < Alt><F> simultaneously.

Window for creating a menu.



Main Heading
Add a new category following those existing.

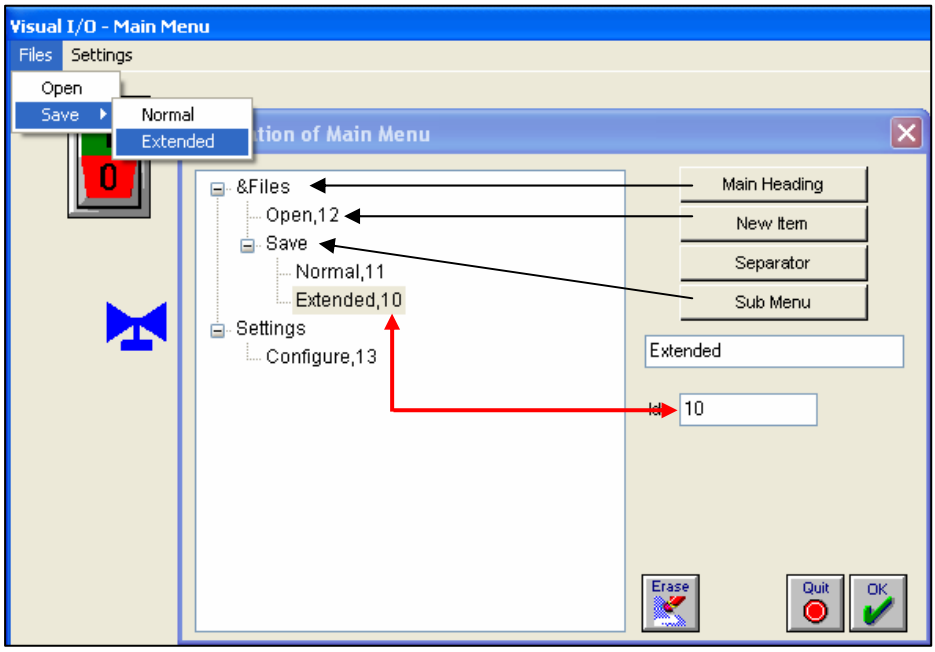
New item
Add item after the current position.

Separator
Add a separator after the current position.

Sub menu
Transform item into new heading (submenu).

At each modification or adding, the main menu of your form is updating.

Example of a main menu



&Files is a main category. The characters **&** before the letter **F** underline it and determine ALT +F keys as shortcut keys.

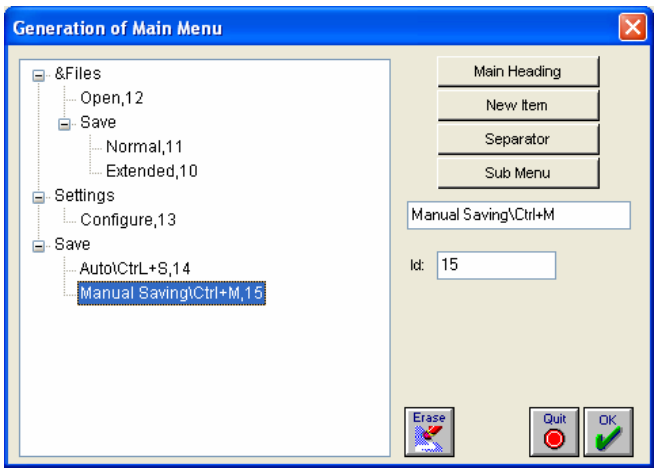
The headings do not have identifiers. Only the text is modifiable in this dialog box.

Open,12 is an item where the text is "Open" and its id value equals 12. This identifier will be tested by the user's program or by a component to determine the action on the menu that will be done.

Save is an item transformed into a sub menu by the button Sub Menu

The identifiers can be modifiable by an edit box

Texts Alignment

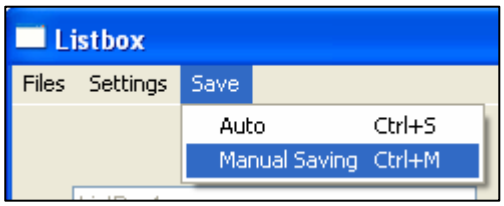


It is possible to align a text completely on the right of menu by inserting a mark (the character \) before this text.

<u>Example</u>

Auto\Ctrl+S

Manual Saving\Ctrl+M

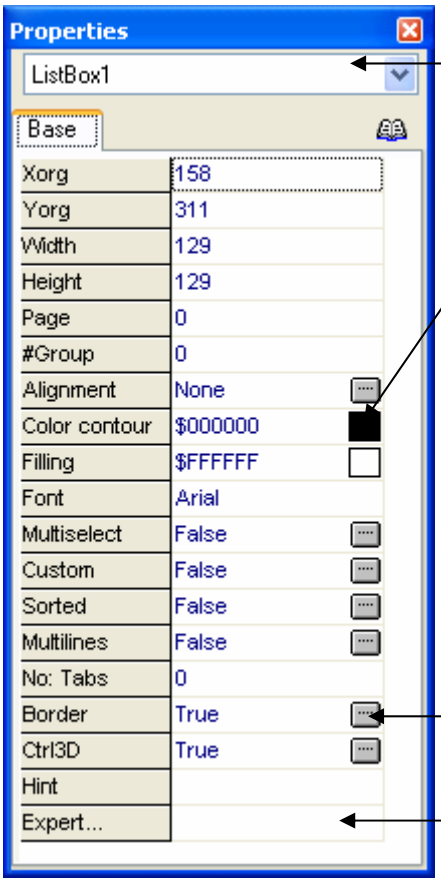


☞ <u>Attention</u>: don't use the same item identifiers twice in an application.

Graphic properties dialog box

The properties dialog box of a shape is called by selecting the shape (click above!). You can strike the **<F6>** key. A window appears:

List of the properties



ComboBox contains the list of all the components present on the form. Choose a component to display its attributes. A palette of customizable color is enabled to set the contour and fill color of a figure. The system colors can be used.

In the case of certain figures, additional properties can also appear following the basic properties.
To call the **current palette**, double click on the square by the **left button** of the mouse.
To display the **Windows' palette**, double click on the square with the **Right button** of the mouse.

For the standard objects window type (listbox, editbox, window) the contour (border) and type of contour (3d, Normal) can be modified.

You can also call the expert while clicking on this cell.

Xorg,Yorg	Coordinates of the top left corner of the selected shape.
Width, Height	Width and height of the selected component.
Color contour	Color contour of the shape. Double click to display a palette of colors (Double click with left or right button of the mouse for display different palettes).
Contour Style	Style of layout of the shapes inherited from the point (lines, Rectangles).
Pen Width	Pencil Width.
Filling	Filling color of the component. Double click (**left or right**) to display a palette of colors.
Filled	Shape displayed, filled or unfilled. Double click to set it.
Font	Text Font. Double-click to call the window for available fonts.
Angle	Modify angle of the text(TT Font only).
Page	Modify page index containing the selected shape (for compatibility only).
No: tabs	Fix the number of tabs in a **PageControl** or a pageSwitcher.
#Group	In the case of the shape is grouped with others, the number of the group allocated by Visual IO appears here. If the shape is not grouped, the number is 0.
Message	Text or Button text. Bitmap, Metafile or Jpeg file name to be displayed. Double Click to display a window for selection and previsualization..
Decimals	Number of decimals for numerical value (V) or number of pages for page switcher.
Multilines	Used exclusively for Edit Box. Indicate whether the Edit Box is multilines or single line.

.

H_Scroll Horizontal Scroll Bar for multilines edit box or **left justification** for a simple **Label**.
If this option is true during the modification of the label by program, this one will always be aligned to the right. If not, the text is justified to the left.

V_Scroll Vertical Scroll Bar for multilines edit box.

Password If this option is true, the content of the edit box is replaced by characters *****.

Multiselect Allow the multiple selections in a list Box or not.

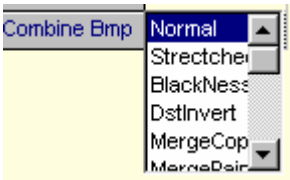
Type Combo Box Type (1 : Simple, 2 : Drop Down, 3 : DropDownList).

Personalized Allow the customization of a List Box or a Combo Box. For example, personalization allows displaying another thing than text in a Combo Box or a List Box. DriveComboBox allows the list of the disk drives in your PC with bitmaps and texts. Use **AddString** procedure with a special formatted text.

Sorted Sorts a List Box or a Combo Box. Customized Combo Box and List Box cannot be sorted.

Bitmap properties

Combine Bmp Graphic combinations for the bitmap and the background.
The most used attributes are normal and stretched.



Normal Display the bitmap in its original size.
Stretched Increased Bitmap size to the boundary.
FormSkin Allows placing Bitmap in the background.

This causes to replace the standard window frame of Windows (Skins).

Alignments

Specify how a control is placed relative to its parent in the case of resizing the form at Run and Design time.

The first part allows increasing a shape for adapter with the new size of the form. For example, a window containing a grid need this functionality.
(D-Right, D-Bottom, D-Right/Bottom, None).

The second part allows moving a shape automatically when resizing a form by the borders of this one.
For example, that always allows visualizing a button whatever the size of the form.
(Right, Bottom, Right/Bottom, None).

The third part is allowed to position a graphic element in size and position on top, bottom, right, left or free zone known as Client.
(Altop, AlBottom, AlRight, AlLeft, AlClient).

None : Cancelling any alignment.

To align a control al top, bottom, on the left or on the right of a form or a tab that it remains on this site even if the size of the form, the tab or the component containing this control changes. When the relative is resized, an aligned control also modifies its dimensions in order to continue to extend towards the higher, lower, left or right side of the parent.

An unspecified number of child components of the same parent can have the same value for Align. In this case, they pile up along the edge of the parent. Child controls are piled up in order Z.
This is an advantage because you frequently want a group of controls to be aligned either on their left, right, top, or bottom edge.

Movings

None The component remains where it was placed. This is the default value.
Possibly to display a scrollbar appears on the border of the form.

D-Right In case of form resizing, the object move to stay at the same place and the same distance from the right border of the form.

D-Bottom In case of form resizing, the object move automatically to stay always at the same distance from the bottom border of the form.

D-Right/Bottom Combination D-Right+D-Bottom.

Enlarging

Right The object enlarges by its right border to remain always at the same distance from the right border of the forms.

Bottom The object enlarges by its bottom border to remain always at the same distance from the bottom border of the forms.

Right/Bottom Combination Right+Bottom.

Size and Position

AlTop The control moves to the top of its parent and resizes to fill the width of its parent. The height of the control is not affected.

AlBottom The control moves to the bottom of its parent and resizes to fill the width of its parent. The height of the control is not affected.

AlLeft The control moves to the left side of its parent and resizes to fill the height of its parent. The width of the control is not affected

AlRight The control moves to the right side of its parent and resizes to fill the height of its parent. The width of the control is not affected

AlClient The control resizes to fill the client area of its parent. If another control has already occupied part of the client area, the control resizes to fit within the remaining client area.

☞***Note:*** The shapes with one of these attributes modify their sizes at Run time regarding to resizing of the form by user.

.

Enlargement of objects

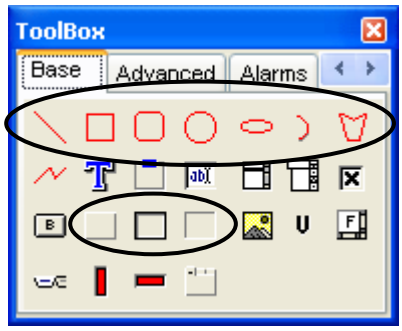
Select the objects
Hit simultaneous <CTRL> <+> for enlarge the selected objects.
Hit simultaneous <CTRL> <-> for reduce the selected objects.

Experts for the graphic shapes.

Expert dialogboxes allow affecting predefined treatments by setting options and information.

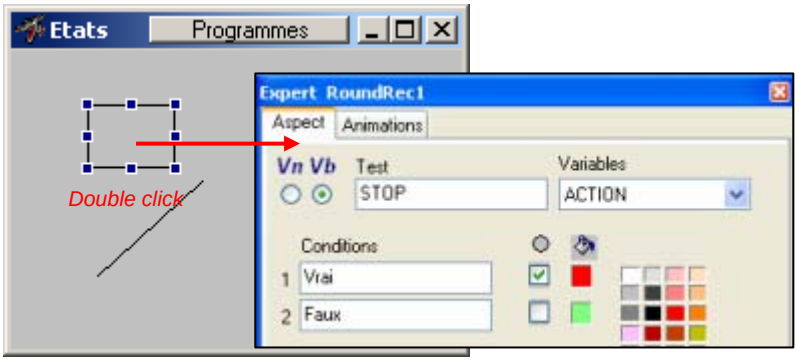
Line, Rectangle, Circle, Ellipse, Arc, Polygon, Supports

Line, Rectangle, Circle, Arc, Ellipse, Polygon, Supports



Expert description

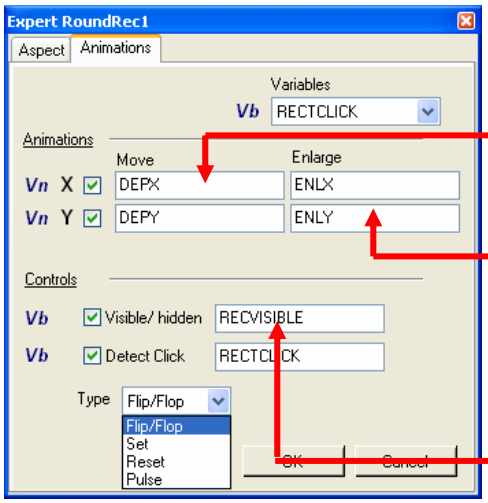
Filling a rectangle according to a bit



Blink Red/Transparent if Stop is true.

Fill in green when Stop is False.

Moving and enlarging a rectangle according to numerical variables.



Moving the rectangle relatively to the original position.

Moving in X according to the variable **DepX** (in pixels).
Moving in Y according to the variable **DepY** (in pixels).

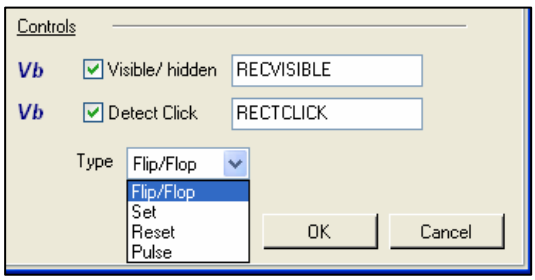
Enlarging a rectangle relatively according to the original size.

Enlarging in X according to the variable ENLX (in pixels).
Enlarging in Y according to the variable ENLY (in pixels).

The Rectangle is visible or hidden according to the Boolean variable RecVisible.

RecVisible=If true, the circle is visible on screen, else it is hidden.

Creating a button with a rectangle



By clicking the rectangle, it is possible to transform it into a button of the Pulse type, reset, Set or Flip/Flop.
The bit RectClick is affected according to the selected type.
If Flip/Flop is the type chosen, a left click with the mouse on the rectangle, set the bit RetClick to True, a second click reset the bit RetClick (False).

Filling a rectangle according to a numerical value

◄ The other type of test is a numerical compare.
Check the radio button Vn, eight tests of numeric variable are possible.
With each test a specific color is defined.
If the condition is true or higher than the value, the rectangle is coloured and can also blink by checking the check box *blink.*
Here variable DEPY is compared with the values 10, 20, 30 or W0 in order to color consequently the round rectangle.

☞ **Note** Only for the **Polygon** or **Polyline**, the rotation is accepted. This rotation is performed according to a given angle contained in the specified variable.

The rotation turns clockwise, which makes the representation of needles in meters easier.

An edit box appears only for these two types of drawings.

Testing a bit in a word

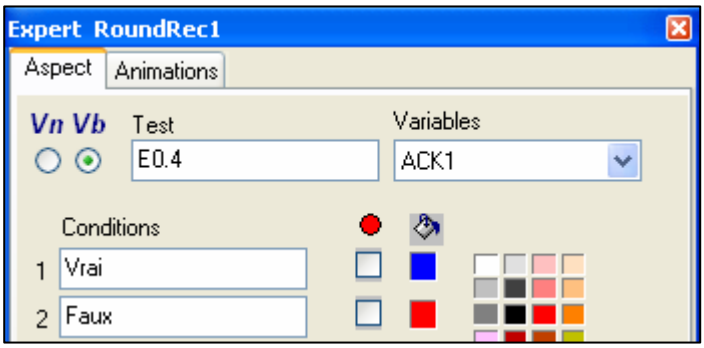
With Visual I/O it is very easy to set or reset a bit in a word, byte or integer.
The nomination by point follows by the desire bit.

The first bit is 0 and the last is the Bit 31.

Example Test of Bit 0 of the Word W10 `If W10.0 Then` (True or False as a Boolean).

Example: Affectation d'un of a bit in a word:

```
W10.4       :=True;
COUNTER.9   :=False;
```



Test of bit 5 in word E0 if this bit is true (1). Fill the rectangle in blue, else in red.

Programming on the rectangles, circles, lines and polygons

```
{*** RoundRec1 ************************************}
SObject Procedure RoundRec1;
Begin
  //*** Programs concerning the Rectangle RoundRec1
End;
```

Change filling color according to a bit.

```
If Stop Then SetFillColor(255) ;  //** Color in Red if stop is true
If Stop=False Then SetfillColor($00FF00);//* Color in green if stop is False.
```

Change contour color according to a bit

```
If Stop Then SetColor(255) ;    //** Color in Red if stop is true
If Stop=False Then SetColor(0); //** Color in black if stop is False.
```

Test of click mouse in the rectangle and display of the name of the component

```
If Toplbdown Then Message('Rectangle ='+GetFigname); //** Message in Sysplc.pas
```

Show and hide according to a bit

```
If RecVisible Then ShowControl
              Else HideControl;
```

Test if the mouse is in the figure

```
If Is_Me Then W0 :=W0+BT100; //** BT100=1 all 100ms
```

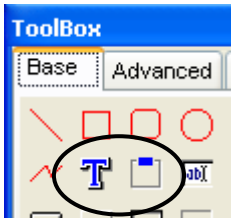
Test if the mouse arrives in the component

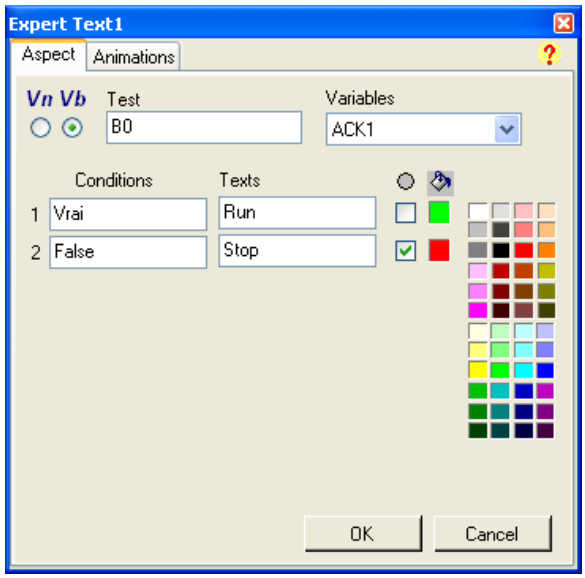
```
If Arrive then
Begin
  Go2('Text1');          //** Change object go to text1
  SetText('Arrive');    //** Change text of figure Text1
End Else
If Depart Then
Begin
  Go2('Text1');
  SetText('Depart');
End;
```

Save the coordinates of a component

```
//*** RoundRec1 ************************************
SObject Procedure RoundRec1;
Var
  Xorg, Yorg     : Integer;
  Width, Height : Integer;
Begin
  //*** At the first display *****
  If Start_Visu Then
  Begin
    Xorg  :=GetXorg;
    Yorg  :=GetYorg;
    Width :=GetXFin-Xorg;
    Height:=GetYFin-Yorg;
  End;
  //***** Width & Height X2 *****
  If Doenlarge then
  Begin
    SetXfin(Xorg + (Width*2));
    SetYfin(Yorg + (Height*2));
  End;
End ;
```

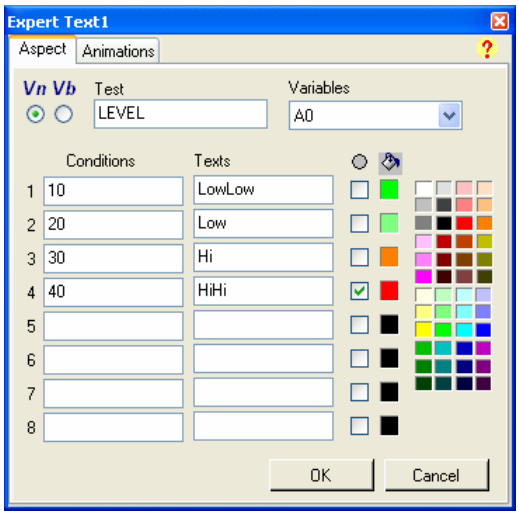
Text, Group





This expert is enabled to change the text, its color and the moving on the form.

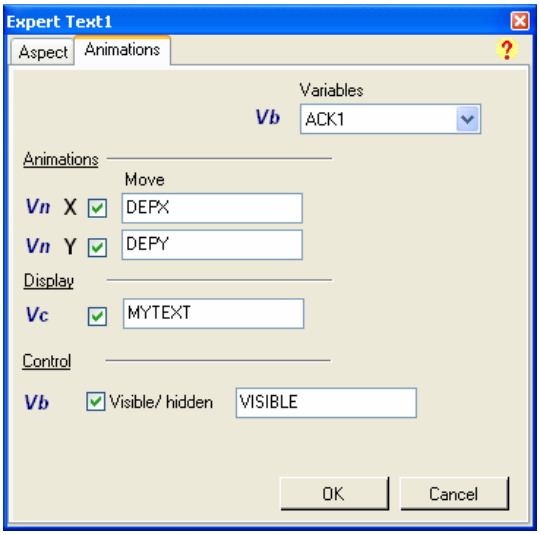
2 types of treatment are possible:
Test of a Boolean variable with 2 possible conditions **True or False**.
Depending on the state, the text will be colored in the specified color and the new label will be displayed. If the check box (below blinking circle) is checked, the label will blink when the condition will be true. Bring by drag and drop, the variable from the ComboBox *Variables* to the edit box *Test* or hit directly the name of the variable to test.
If this variable does not exist, Visual I/O will create it automatically for you by adding it at the end of the global variables list.



The other type of test is a numerical comparison.
The variable tested is not a Boolean variable but a numerical value (Word, Integer or Real).
8 Conditions are possible.
For each condition a new color and a label affect the text that is blinking or not.

The condition to test can also be a simple numeric variable.

Animations Text, Group



According to the edit box focused, the list of the global variables changes Combo '*Variables*'.

For the Animation section, only the numerical variables are allowed.

The field move in X or Y indicate the variable containing displacement relating to the origin coordinates.

Ex: If the DepX variable contains 10, the text will move of 10 Pixels in X from its original position drawn at design time.

The Display option displays the contents of a string variable on the form. Here the text on the screen will write constantly the contents of the variable MyText.

The **Visible/Hidden** option shows or hides the component according to the state indicated by Boolean variable. Ex: Here **if** visible is True **then** the text will be visible in the form **else** it will be hidden.

☞ **Note** : The group object does not have any functionality, it is simply one rectangle including a text.

It has a function to surround other objects. It is always imperative to trace the group before the objects that will be surrounded. On the contrary, Visual I/O will detect the click mouse on the group and never on the objects inside this one.

Text on several lines

Insert character # in the character string, so that the system inserting soft carriage returns text wraps at the right margin. Character # is by all means replaced by a space character. The text is wrapped horizontally.

Example:

ARSOFT#International Display under visual I/O:

Programing

In case that the expert is not used, you can change the text by the **SetTextMessage** procedure.

Changing the text if the mouse move over the object.

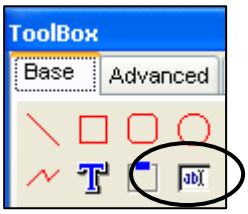
```
// *** Text1 ************************************
SObject Procedure Text1;
Begin
 If Is_me Then SetTextMessage('I say Hello')
          Else SetTextMessage('you say goodbye');
End;
```

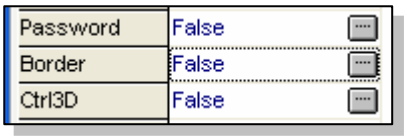
Changing font name when the mouse move over the object.

```
// *** Text1 ************************************
SObject Procedure Text1;
Begin
 If Is_me Then Setfontname ('Courier New')
          Else Setfontname ('Arial');
End;
```

NB: **SetText** et **GetText** also run and can replace **SetTextMessage** and **GetTextMessage**.

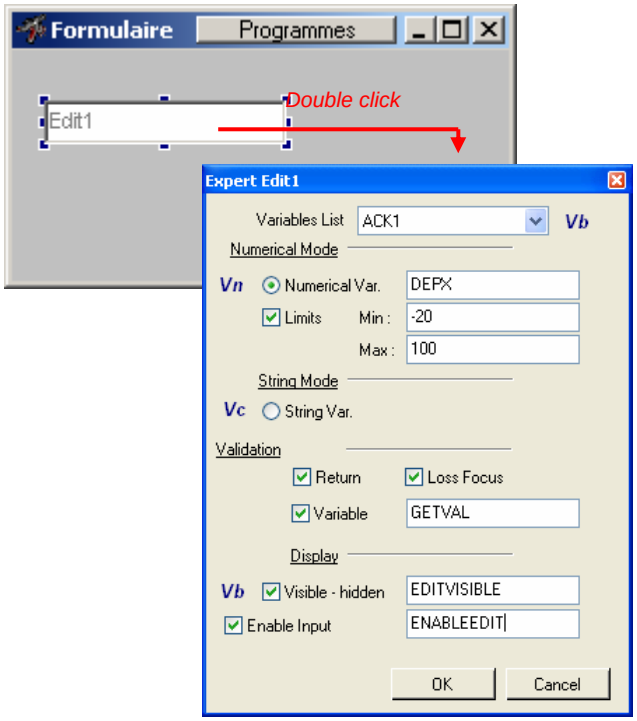
EditBox





Hit <F6> to display the properties box.
Border false makes it possible to hide the contour of the editbox.
Ctrl3D allow modifying the aspect of the contour.
Password option creates an **editbox** that displays a special character in the place of any entered text (character *).

Acquisition of a numerical value



Depx: receives the numerical value entered.
Min: Minimum authorized value.
Max: Maximum authorized value.
The Depx variable receives the entered value if:
The operator presses the <Enter> key
Or
The edit box is left by the <TAB> key
Or
The mouse is moved in another Edit Box.
Or
Variable GETVAL is set (True).

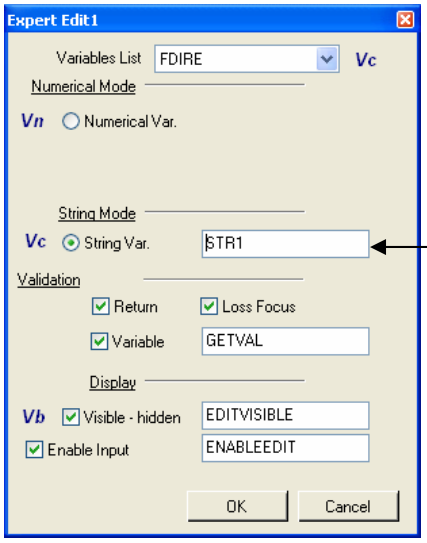
☞ **Note** : Validation can be one of the conditions or a combination of these 3 conditions.

To enable editing in an edit box

Visible - Hidden: If this variable is false, the edit box is hidden. Otherwise the edit box is visible.

Enable Input: Used to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard.

Entering a string



The variable string STR1, follows the same rules as the acquisition of a numeric variable. The characters entered by the operator will be sent in this variable in String type.

If no validation option is check, each character enters directly in STR1 variable.

Programming on Edit Box.

Change background color when the edit box has the caret

```
SObject Procedure Edit1;
Begin
 If Get_Focus Then SetFillcolor($80FFFF) //If editbox has the focus yellow color
              Else FirstColors;     //*** if no focus return to original colors
End;
```

Test the <Enter> key

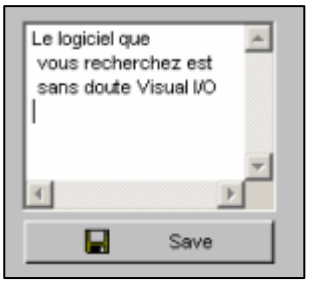
Test the editing of a numerical value

```
{*** Edit2 ****************************************}
SObject Procedure Edit2;
Var
  Status : Integer;
  R      : Real;
Begin
  If EditCR then //*** the < Enter > key is pressed?
  Begin
    Val(GetText,R,Status);    //*** GetText retrieves the text hit
    If Status<>0 Then Clear; //**** Val converts the value into reel if status=0.
  End;
  If EditEsc Then SetText('0'); //*** Test if < Esc >key is pressed then reset
End;
```

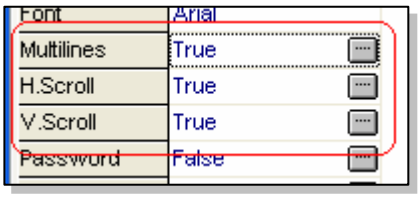
Passing to the next editbox

```
Uses Windows;  //*** Windows.vpu contains the declarations VK_F1, VK_F6 etc..
//*** if <F6> then go to the next editbox, this line can be written everywhere in
//** the form.
If Getvirkey=VK_F6 Then NextFocus;
```

Save the content of an editbox into a text file

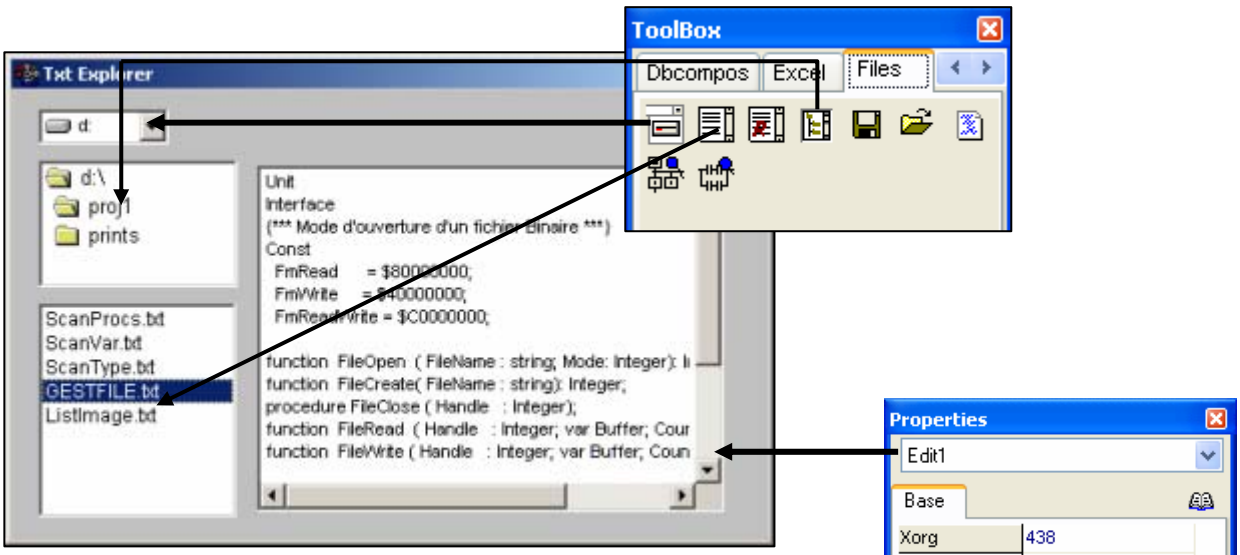


The editbox is multi lines, thus authorizing the editing on several lines. Vertical and horizontal scrollers are possible while specifying in properties box.



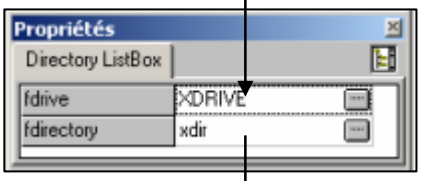

```
Uses GestFile ; //*** for using the DirectoryCreate function
{*** Button2 ****************************************}
SObject Procedure Button2;
Var
  Fic : Integer;
  I   : Integer;
  S   : String;
Begin
  If Button Then   //*** Click on Save button ?
  Begin
    Go2('Edit4'); //*** go to the multilines editbox
    If DirectoryExist('D:\Tempx\Doc')=False Then //* if the directory doesn't
    DirectoryCreate('D:\Tempx\Doc');             //* exists we create it
    FileTextAppend(Fic,'D:\tempx\Doc\Readme.txt'); //* open the file or create it
    For I:=0 To GetNumLines-1 Do
    Begin
      GetLine(S,I);   //*** get each line
      Writeln(Fic,S); //*** write at the end of the file
    End;
    FileTextClose(Fic); //*** close the file
  End;
End;
```

Display the contents of a text file in an edit box

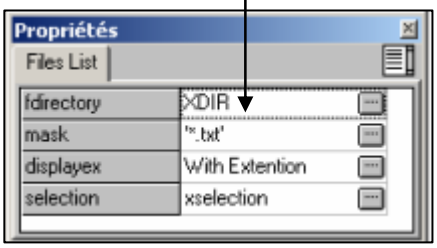


Component **Drive Combobox**

Component **Directory ListBox**



Component **Files List**




```
{*** Edit1 ****************************************}
SObject Procedure Edit1;
Var
 Fic : Integer;
 TS  : String;
Begin
 //*** Test if the string value changed and not empty
 If StrChange(XSelection) and (XSelection<>'') Then
 Begin
   Clear;                          //*** clear the editbox
   FileTextOpen(Fic,XSelection);   //** open the file
   hidecontrol;                    //** Hide the editbox (to be faster)
   While Not(Eof(Fic)) Do          //** scan all the file
   Begin
     Readln(Fic,TS);    //** read line per line
     Addstring(TS);     //*** add the read line
   End;
   Showcontrol;           //*** show the content of the editbox
   FileTextClose(Fic);    //*** Close the text file
 End;
End;
```

Note :
Instructions *HideControl* and *Showcontrol* is enabled to hide the editbox temporarily for filling it with lines read in the file. These 2 instructions can be eliminated. Thus the operator will see the editbox filling line per line.
This code also can be used behind a normal listbox.
The result will be identical except that the operator will not be able to modify the lines.

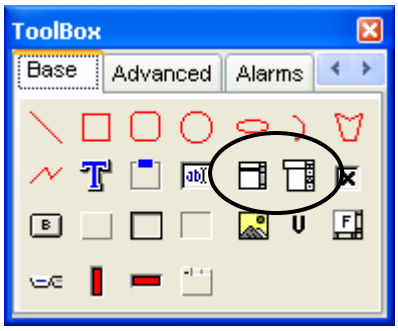
Drag & drop from an editbox to another.

```
// *** Edit2 ************************************
SObject Procedure Edit2;
Begin
  Drag(GetText); //** Drag make it possible to get the text return by GetText
End;
// *** Edit3 ************************************
SObject Procedure Edit3;
Begin
 If Drop Then  //** Drop indicates if the operator release the drag & drop in the object
   Settext(Getdrop); //** if yes forcing of the editbox with the text of Drag
End;
```

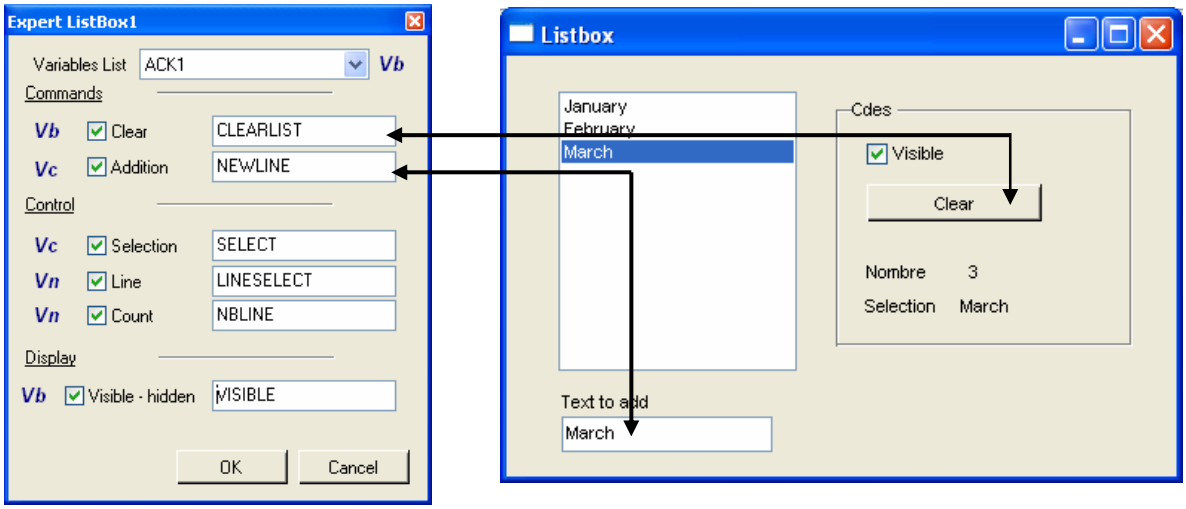
Select the contents of an editbox

```
// *** Edit3 ************************************
SObject Procedure Edit3;
Begin
  If B1 Then SetSelection(0,255); //** Selection from first to 255
End;
```

ListBox, ComboBox



The expert



Clear: clears the contents of the Listbox when this variable is True.

Addition: Is a String variable. When its contents change the string, it is added to the end of the list. Otherwise, the string is inserted into the list and the list is sorted (see properties).

Selection : Is a string variable, retrieves the list item currently selected in A String (video reverses in ListBox).

Line : Is a numeric variable, which indicates the number of the line pointed (video reverses in ListBox). The first line is number 0. If no line is presents in the ListBox, this variable is set to -1

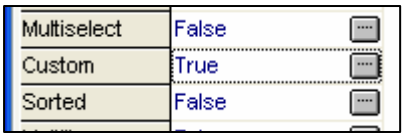
Number : Is a numeric variable retrieving the lines count in the ListBox.

The **Visible/Hidden** option allows showing or hiding the control according to the state indicated by Boolean variable.
Ex: Here if Visible is True then the shape is visible on the form, else the shape is hidden.

ComboBox functions have the same functionalities as the ListBox

Case of a customized listbox or combobox

It is possible to display bitmaps, texts in colour and the texts aligned differently.
You need to set the listbox in owner draw.
Hit **<F6>** then set the attribute Custom to true.



The Addstring procedure can normally add lines into the current listbox and also with others features.

Addstring('**Bitmap**'+#1+'**Text**'+#1+'**Color**'+#1+<**Shift right in pixels** ***optionnal***'>.

Programming on Listbox.

Initialisation of the content of a listbox or a Combobox.

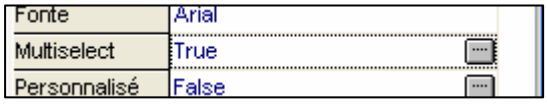
```
IF Start_Visu Then //** at launching time
Begin
  Addstring('Visual IO'); //** add Text Visual IO
  Addstring('Visual PLC); //** add text Visual PLC
End;
```

Send the content of a listbox to the serial comport.

```
Uses Serial;
// *** ListBox1 ************************************
SObject Procedure ListBox1;
Var
 I : Integer;
Begin
  If Go Then
  Begin
    Opencom('1','9600','E','8','1'); // ** open the serial comport
    For I:=0 to Getcount-1 Do        // ** scan all lines
    Begin
      Comwrite(1,Getstring(I)); // ** send the line on COM1
    End;
    Disablecom(1); // ** Close the serial com COM1
  End;
End;
```

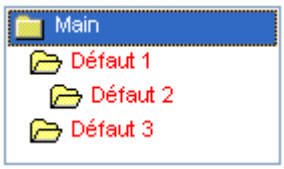
Send the selected lines of a listbox to a file.

Hit **<F6>** in the properties box, you must **Multiselect** at **true**




```
// *** ListBox1 ************************************
SObject Procedure ListBox1;
Var
 Fic : Integer;
 I   : Integer;
Begin
  If Go Then
  Begin
    Filedelete('Temp.txt');             // ** Clear the final file
    Filetextappend( Fic,'Temp.txt');    // ** Open the file to add
    For I:=0 to Getcount-1 do           // ** Scan all the listbox
    Begin
      If Getselected(I) then            // ** if the line is selected
         Writeln(Fic,Getstring(I));     // ** we send it to the file
    End;
    FileTextClose(Fic);     // ** Close the file
  End;
End;
```

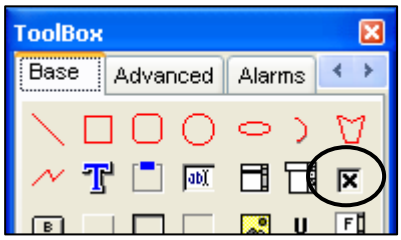
Add strings in a customized listbox.

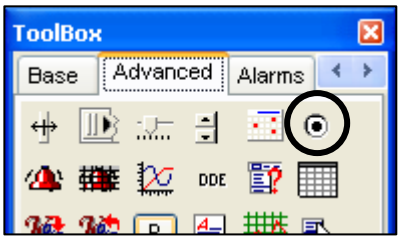


Display a bitmap with shift.
The functions above also run in customized mode.
Note: The function **VisuBMP**(Line Num) returns the bitmap name at the start of the line

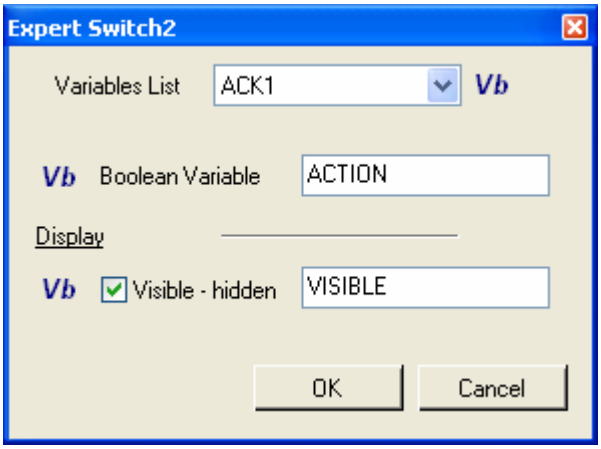
```
// *** ListBox1 ************************************
SObject Procedure ListBox1;
Begin
  If Go Then
  Begin
   Addstring('VPCLOSEDFOLDER'+#1+'Main'+#1+'$FF0000'+#1);      // no shift
   Addstring('OpenFolder'+#1+'Defaut 1'+#1+'$0000FF'+#1+'10'); //shift of 10 pixels
   Addstring('OpenFolder'+#1+'Défaut 2'+#1+'$0000FF'+#1+'20'); //shift of 20 pixels
   Addstring('OpenFolder'+#1+'Défaut 3'+#1+'$0000FF'+#1+'10');
  End;
End;
```

Checkbox and Radio button





The checkbox and the radio button affect the state of a Boolean variable on changing state.



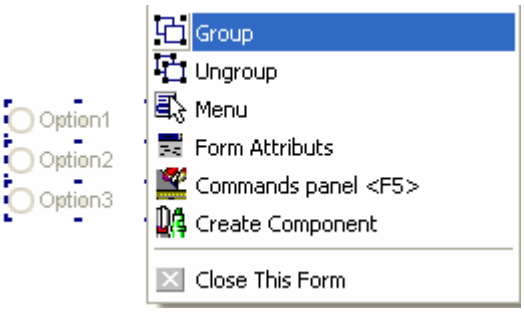
◄ **Boolean variable** retrieves the state of the checkbox. If checked, the Boolean variable is set. Otherwise, the Boolean variable is reset.

The **Visible/Hidden** option is allowed to show or hide the control according to the state indicated by the Boolean variable.
Ex: Here if Visible is True then the control is visible on the form, else the control is hidden.

Radiobutton

The Radio Button also is enabled to assign a binary state (0 or 1) to a Boolean variable.
A Radio-Button clicked passes in the checked state can be unchecked only by one click on another radio button.

So the Radio operator Buttons is a mutual exclusion. They should be grouped in the editor of Visual I/O.
With this intention, select the Radio Buttons which have a mutual exclusion, then group them by the order of grouping them in the popup menu (right click).
Thanks to this method, it is possible to define groups of Radio Buttons.



◄ Here the example of grouping the Radio buttons Option 1,2 and 3 altogether, those will work in mutual exclusion.

Programming on checkboxes and radio buttons.

Test if the state of a check box has changed

```
// *** Switch5 ************************************
SObject Procedure Switch5;
Begin
  If ISModified Then A5:=A5+1; //** if the state change, Ismodified is set
                               // only on one cycle
End;
```

Check or uncheck a checkbox or a radio button by program

```
// *** Switch5 ************************************
// ** Works right on radio button or check box
SObject Procedure Switch5;
Begin
  If Go Then Check // if the variable Go=True, check the checkbox or the radiobutton
         Else Uncheck; // ** if the variable Go=False, uncheck the checkbox
End;
```

☞ **Note**: If this program is applied to a radio button, the uncheck action uncheck automatically the other radio buttons because these are in mutual exclusion. Concerning the radio button the procedure SetRadio (State: Boolean) can be used in the check and uncheck procedure.

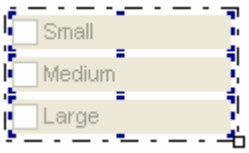
Test the state of a check box

```
// *** Switch5 ************************************
SObject Procedure Switch5;
Begin
  If (GetCheck=1) And (IsModified) Then A5:=A5+1; //** if the state has changed
                  //and the check box is checked then inc the word A5
End;
```

Test the state of a Radio Button

```
// *** Switch5 ************************************
SObject Procedure Switch5;
Begin
  If (GetRadio) And (IsModified) Then A5:=A5+1; //** if the state has changed
                  //and the radio button is checked, inc. the word A5
End;
```

Reset all check boxes in a group

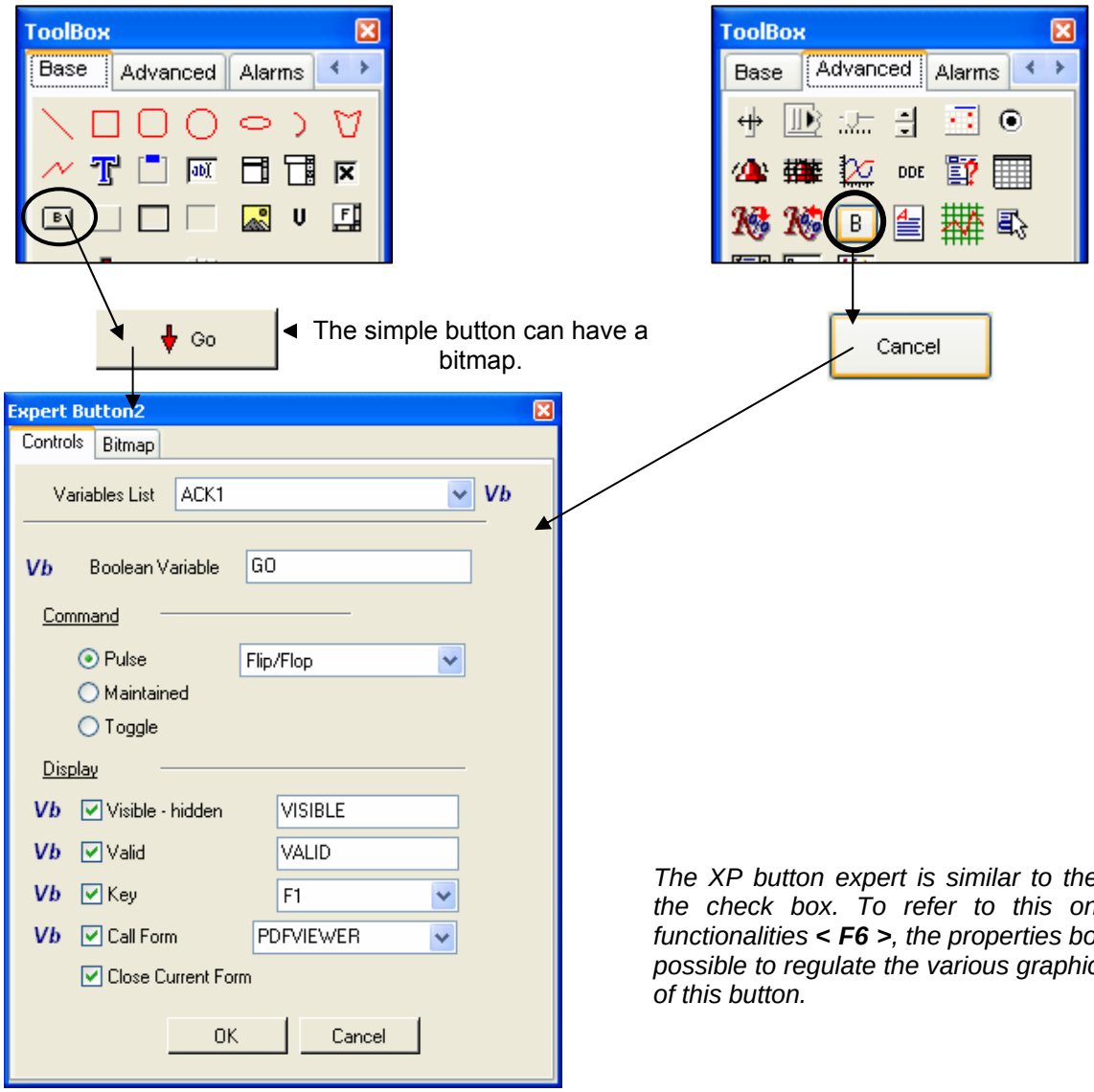


◄ The check boxs must be grouped. If the check box Small is the traced first below (Head of the group), the program must be written behind.

```
// *** Switch5 ************************************
SObject Procedure Switch5;
Begin
  If Go Then
  Begin
    Repeat          // ** Start on Small
      Uncheck;      // ** Uncheck the current check box
    Until Next_Group=False;  // ** go to the next draw or return false
  End;                       // ** if it is the last draw
End;
```

Button

Two types of buttons are available:
The simple button in the Base tab which is managed entirely by the Visual I/O system.
The button XP is typed in the Advanced tab and managed by Windows XP.



*The XP button expert is similar to the expert of the check box. To refer to this one for the functionalities **< F6 >**, the properties box makes it possible to regulate the various graphic attributes of this button.*

Boolean variable retrieves the result of the action on the button.

The button can be :
Pulse: The variable is set (true) only on one cycle of program, otherwise returns false.

Flip/Flop: Change state of Boolean variable on each click.

Reset: Reset the Boolean variable to False (0).

Set: Set the Boolean variable To True (1).

Maintained: Variable is set to True (1) as long as the button is on.

Toggle:
When clicking, the button is set and remained set. It affects the Boolean variable to True (1) as long as this one is set. A second click changes the state of the button and passes the Boolean variable to False (0).

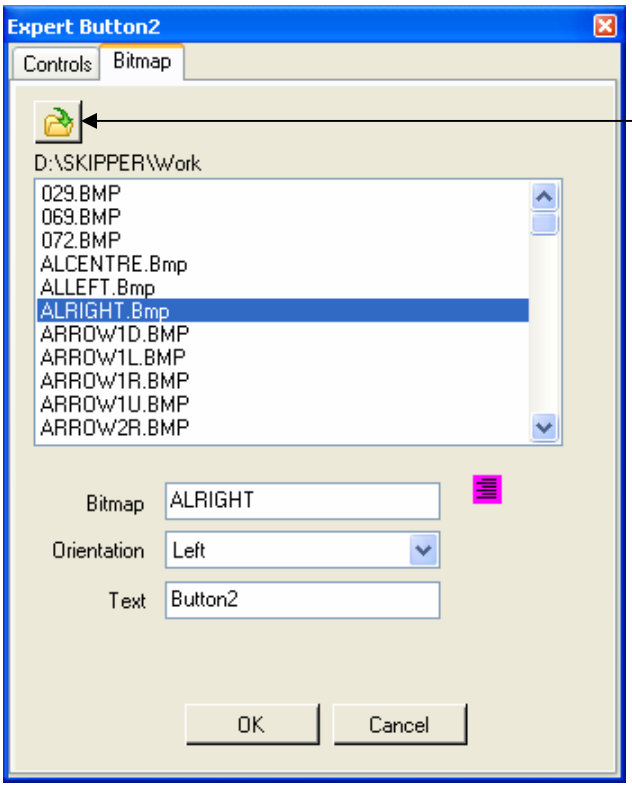
Visible/Hidden: Show or hide the Button according to state of indicated Boolean variable.

Valid: Enable (grey Button) the button according to state of Boolean variable. If this variable is False, the button is grayed. A click of the mouse has no effect on this one.

Key: Allow attaching an alphanumeric key to this button.

Call Form: enable calling a form by clicking the button.

Close current form: Enable to close the synoptic containing this button. If the synoptic is main (First to be displayed when launching of the application), the complete application is closed.



Bitmap: Allow displaying a small Bitmap next to the text of the button.
To select a Bitmap, the navigation is possible by clicking the *open button*.
The selected bitmap is then copied into the current project.
Bitmap must consist of 2 parts. First part representing *Valid* Bitmap and another part *inactive* Bitmap (Grayed). This corresponds to the 2 possible states of the button. Bitmap 32x16 is a correct dimension. A list of buttons is available in installing directory (/Buttons).

Orientation of Bitmap, allows positioning Bitmap on the left, Right, Top & Bottom.

☞ **Note** :

Bitmap of Width=Height is also possible. However, this one will not have a grayed aspect.

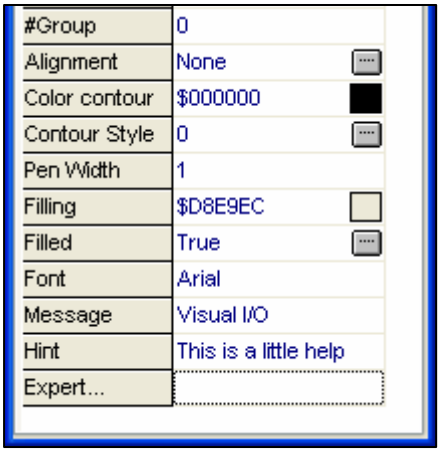
Text of a button on several lines

If the button is too small to display the text, this one can be written on several lines, the second line begins at the first character space found.

◄ Here the button is enough large to display the text.

◄ Here the button is not enough large to display on his length. The text is cut at the level of space character.

Properties box



#Group	0
Alignment	None
Color contour	$000000
Contour Style	0
Pen Width	1
Filling	$D8E9EC
Filled	True
Font	Arial
Message	Visual I/O
Hint	This is a little help
Expert...	

Filling allows setting the color of the button.
Contour Color allows fixing the text color of the button.

Filled = False makes it possible to create a flat button without contour.
The contour appears when the user pass the mouse over.

Hint: To help all objects a little Window when the mouse runs over the button.

Programming on buttons

Attention to the Button function which is valid only if the expert does not contain any Boolean !

Test if a button is clicked

```
If clicked, the text of the button is changed by the "Stop" text
// *** Button1 ************************************
SObject Procedure Button1;
Begin
 If Button then SetText('STOP'); //** the function button returns True on 1 cycle
End;
```

Change the button color according to a numerical value

The colors ClRed.. are defined in Windows.vpu then it is necessary to write **Uses Windows.**

```
Uses Windows;
// *** Button1 ************************************
SObject Procedure Button1;
Begin
 Case A1 of
      1 : Setfillcolor(clRed); //* Setfillcolor colors the interior of the button
      2 : Setfillcolor(clBlue);
      3 : Setfillcolor(clGreen);
      4 : Setfillcolor(clOlive);
      5 : Setfillcolor(clPurple);
      Else Firstcolors; // ** Returns to the original colours if A1<>12,3,4 or 5
 End;
 SetColor(CLWhite XOR GetFillColor); // Make the text always visible
End;
```

Change the button color when the mouse arrive on it

```
// *** Button1 ************************************
SObject Procedure Button1;
Begin
  If Arrive Then Setfillcolor(ClRed);//When the mouse arrives,color is set to red
  If Depart Then Firstcolors;    // When the mouse leaves, the button return to
End;                            // the original colors
```

To hide a button according to a Boolean state

Enable and disable the functionalities according to external conditions.
This functionality is also available automatically in the expert button.

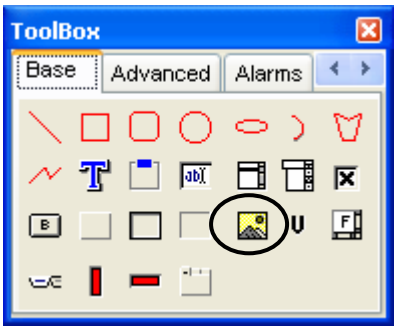
```
// *** Button1 ************************************
SObject Procedure Button1;
Begin
 if Action Then Showcontrol
           Else Hidecontrol;
End;
```

Change the font and the style of a button

```
// *** Button1 ************************************
SObject Procedure Button1;
Begin
  if Action Then
  Begin
    SetFontName('Courier');
    SetFontStyle(2); // ** Italic
  End;
End;
```

☞ **Note**: Lines, Circles, Rectangle, support, Ellipse, Polygon, Polylines can also be buttons by the attribute *detect click* in the expert dialogbox.
The function **Button** also runs **on all the graphic figures**.

Image

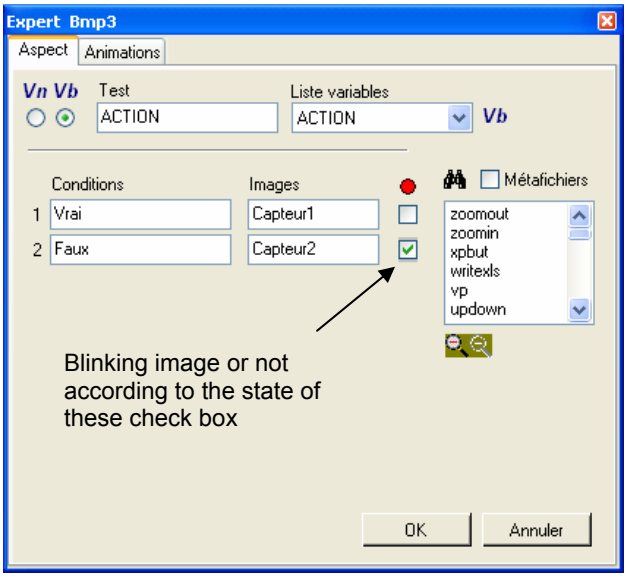


The Image object is allowed to display different formats:

Bitmap	: .BMP
Meta file	: .WMF
Jpeg	: .JPG

These images are included in the final executable file, simplifying the installation of your applications.

Display an image according to selected conditions. This expert is very close to the text one.

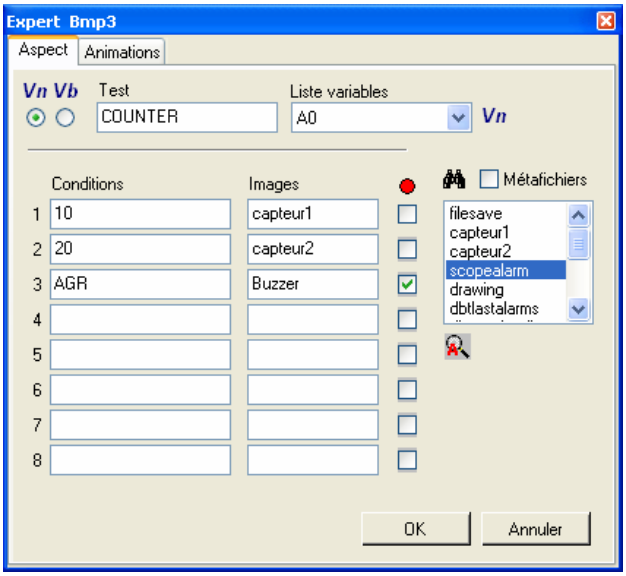


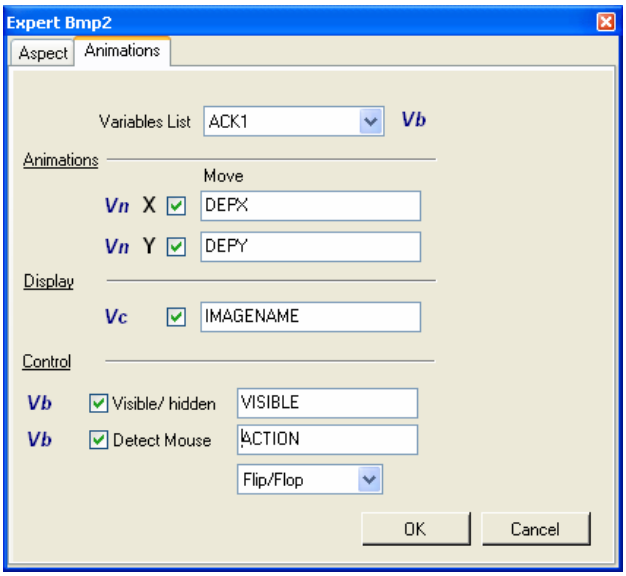
2 types of functions are possible: Testing a Boolean variable with its 2 conditions True or False. According to this state, the image name set will be displayed. If the check box (below blinking circle) is checked, the image will blink until the condition is true. Bring the chosen variable from the combobox Variables to the test field. Or strike the name of the variable to be tested. If this variable does not exist, the system creates it for you automatically and adds it into the list of the variables.
The list of the images of the current project is displayed in the ListBox of right-hand side. The extension of the Images is W

The other test is a numerical comparison. The variable tested is not a Boolean variable but a numerical variable (Word, Integer or Real).

8 conditions are available.

For each condition, an image name is affected with or without blinking.
The condition to test can also refer to another numeric variable.





According to the field, the list of the variables of ComboBox Variables changes. In the 'Animations' section only the numeric variables are allowed. The fields 'move' in X or Y indicate the variable containing moving relating to the original coordinates.

Ex: If the DepX variable contains 10, the image will move for 10 Pixels in X from its original position, which you had drawn under the editor.

DepX: Absolute moving (X axis) from the original position (in design).
DepY: Absolute moving (Y axis) from the origin position (in design)
Display: Allows displaying the image specified in a String variable

Visible/Hidden allows showing or hiding the image according to the state of the Boolean variable.

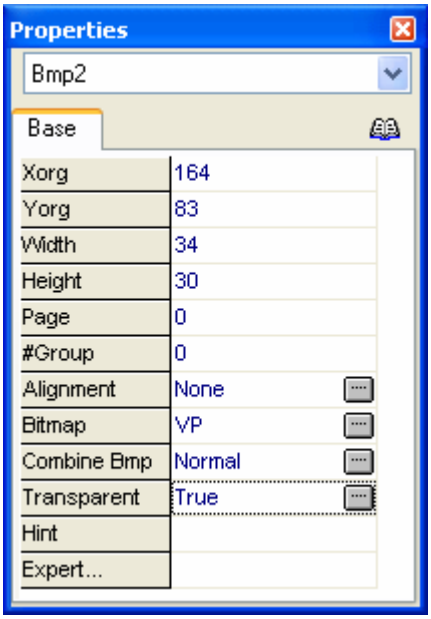
Ex: Here if VISIBLE is True, the image is visible on the form, if False it is hidden.

Detect Mouse For associating a Boolean variable to a Mouse click.
Here if the user click on the image, the variable named ACTION is set.
Several actions are possible by selection in the ComboBox.

Flip/Flop: Invert State of the Boolean variable when the user clicks on image.
Set : Set (true) the Boolean variable on each click.
Reset : Reset (False) the Boolean variable on each click.

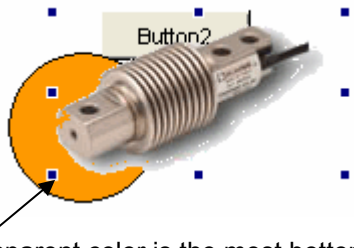
☞ **Note** : The variable character string specified in the field Display indicates the name of the image to be display at run time. This name includes the file extension or not. Visual I/O searches the file in the executable itself. If the file is not found, the application search in the current directory for BMP, WMF or JPEG files.

Transparent Bitmap



The **transparent** option draws the bitmap transparently.
The transparent color is the most bottom left pixel shown on screen.

Combine BMP only runs if Transparent is false and have to combine the pixels of the bitmap with those of the background giving graphic effects.



The transparent color is the most bottom left pixel shown on screen.

Programming on images

Change the image displayed according to a numerical value

```
// *** Bmp3 ************************************
SObject Procedure Bmp3;
Var
  ImageBase : String; // ** This variable belongs to the Bmp3 object
Begin
 If Start_Visu Then ImageBase:=GetTextMessage; //* put the original image name
                                               //  into memory
 Case A1 of
      1 : SetTextMessage('Capteur1'); //** display Capteur1.Bmp
      2 : SetTextMessage('Capteur2'); //** display Capteur2.Bmp
      3 : SetTextMessage('Capteur3');
      4 : SetTextMessage('Capteur4');
      Else SetTextMessage(ImageBase); //** returns to the original image
 End;
End;
```

☞ **Note** : The *ImageBase* variable as other variables is safeguarded and belongs to BMP3 object.

Move on screen the image thanks to the mouse

```
// *** Bmp3 ************************************
SObject Procedure Bmp3;
Begin
   MoveFig; // ** this instruction is enabled to move any object with the mouse
End;
```

Adjust the bitmap to the image frame

StretchBmp does not run when the transparent attribute is true.

```
// *** Bmp3 ************************************
SObject Procedure Bmp3;
Begin
 If action then Stretchbmp(1);
End;
```

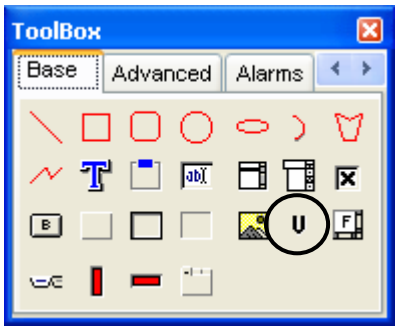
Transfer a bitmap image in another by its handle

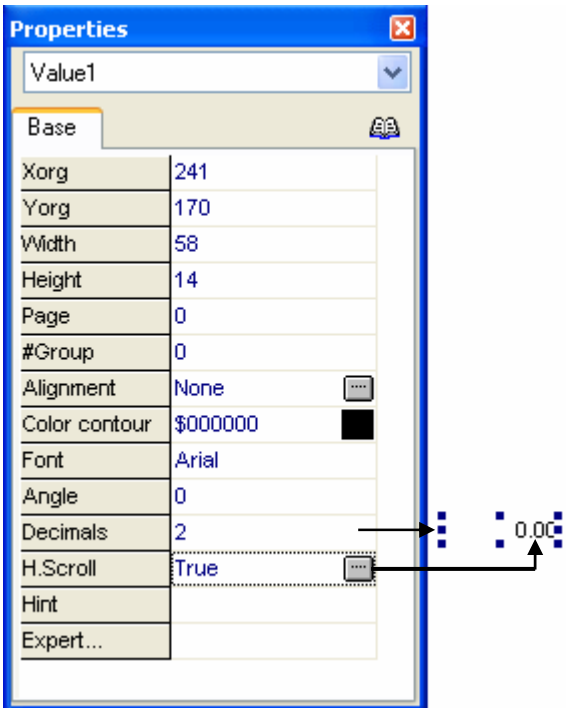
```
// *** Bmp3 ************************************
SObject Procedure Bmp3;
Var
 HD : Integer;
Begin
if action then
 Begin
   HD:=Gethandlebmp;     // ** Get the bitmap Handle of BMP3
   Commut_fig('BMP4');   // ** go to the BMP4 bitmap
   Sethandlebmp(HD);     // ** Copy the BMP3 handle in the BMP4 handle
 End;
End;
```

☞ **Note**: Thanks to the bitmap handle, it is possible to recover all information relating to its size, plane, bitsPixel etc. which use the function Windows **GetObject**. This programming system leaves the framework of Visual I/O referring to Windows SDK.

For example, see the Source of Function **DuplicateBitmap** in **Graphics.PAS**.

Numerical Value



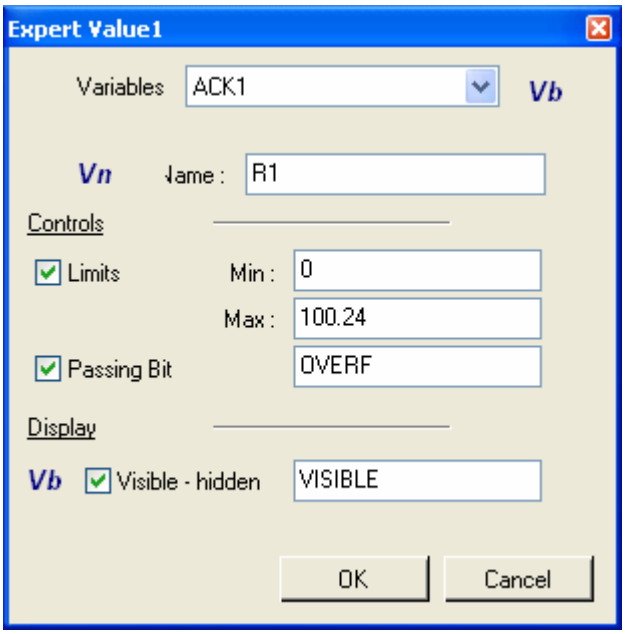


To display Decimals, you can set the number of decimals, call the properties box by striking the **<F6>** key then set the **Decimals** property**.**

To align the text on the right, set the property **H.scroll to true**

Angle makes it possible to display the value under a given angle.

Expert numerical value



The numerical variable to display is contained in the field *Name*.

This variable can be limited. On the other hand, it cannot be out of the specified limits. If this case occurs, the passing bit indicated will be set and the numerical value will be automatically limited at the boundaries.

The PASSING bit is set automatically but it must be reset (False) by program (this bit is a latched Bit).

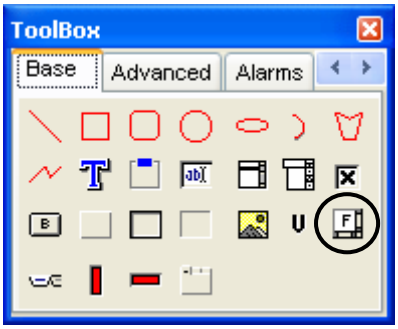
Visible-hidden: allow displaying or hiding the numerical value according to the state of the Boolean variable.

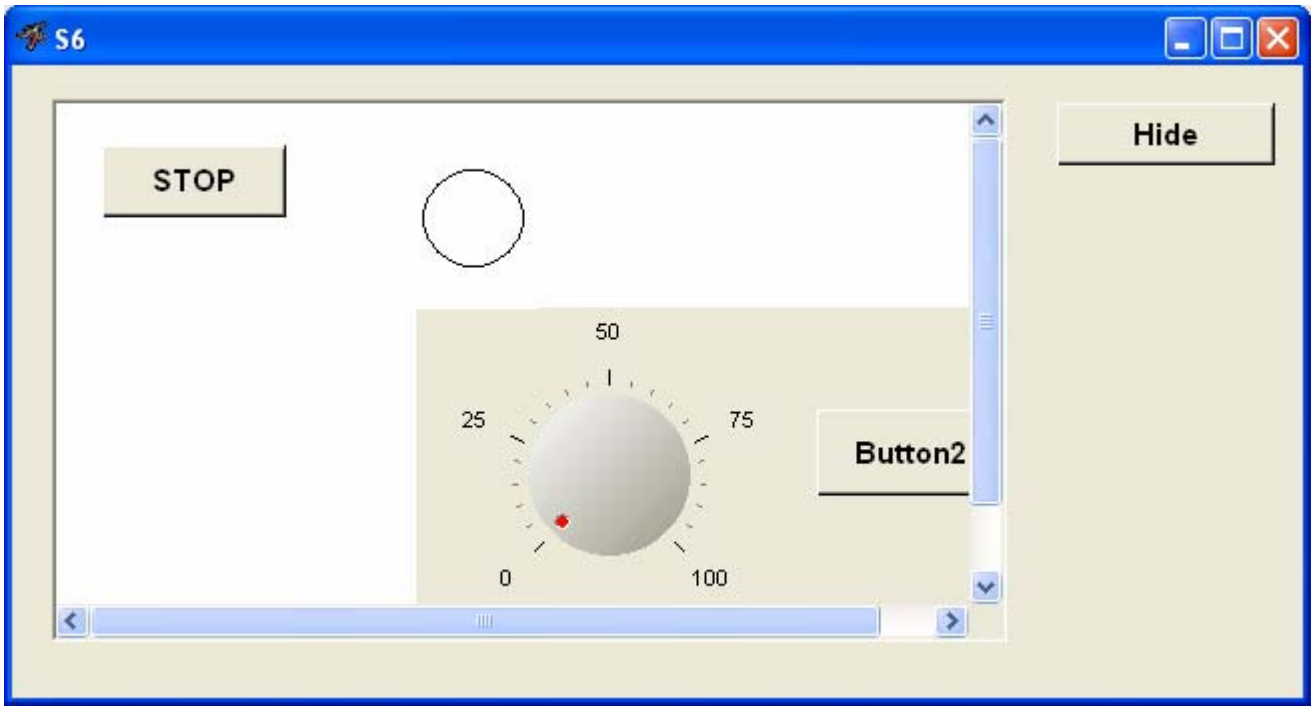
Programming on numerical values

Adjust the number of decimals of a numerical value et sets its value

```
// *** Value2 ************************************
SObject Procedure Value2;
Begin
  If action then
  Begin
    Setndecimales(5);       // *** Set 5 digits after comma
    SetValue(Getvalue+1);   // *** The current value's increment by 1.
  End;
End;
```

Internal window





The internal window has not expert. It is a container allowing displaying shapes.
The scroll bars appear automatically on the scrolling Windows controls if it is not large enough to display all of its controls.

It is possible to show or hide the window by program. Select the window by clicking right button of the mouse, call the program editor :

Programming on internal windows

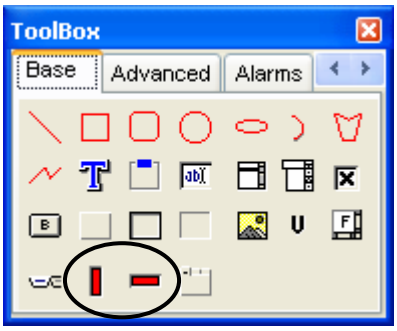
Show, hide and fill the window background

```
Uses Windows;
// *** Window1 ************************************
SObject Procedure Window1;
Begin
 if action then Showcontrol   // ** Display the window
           Else Hidecontrol;  // ** Hide the window
  If B0 Then Setfillcolor(ClRed) // ** colorize in red the background
        Else Firstcolors;        // ** Returns to the original colors
End;
```

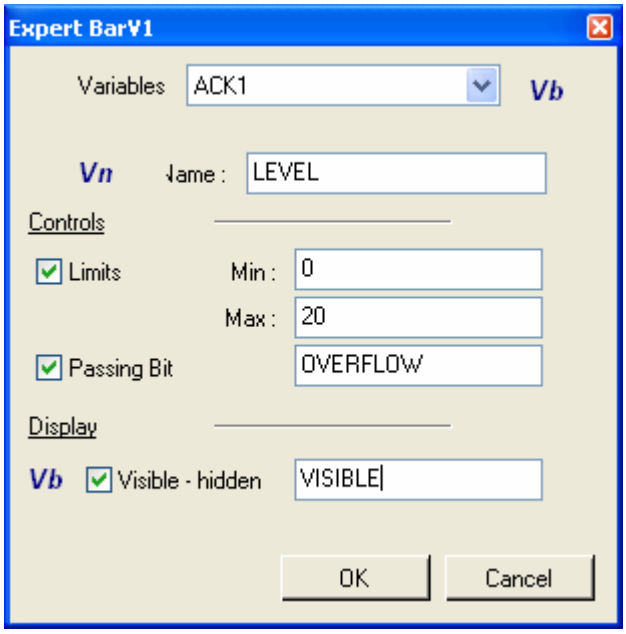
☞ **Note** :

The function **ShowControl** displays any graphical shape.
The function **HideControl** hides any graphical shape.

Vertical and horizontal Bargraph



The Vertical and horizontal bargraph follow the same rules as those of the numerical value.
These 2 bargraphs evolve basic height from 0 (Minimum) to 100 (Maximum), but they can be set to another scale by filling the Min and Max fields.



The numerical variable to display is contained in the field '**Name**'.

This variable can be limited.
On the other hand, it cannot be out of the specified limits. If this case occurs, the specified passing bit will be set (True) and the numerical value will be automatically limited at the boundaries.

The **PASSING bit** is set automatically but **must be reset** (False) **by program** (this bit is a latch).

Min and Max values also indicate the evolution of the Bargraph (here the bargraph will be 0 when the variable level is less than or equal to 10. The bargraph will be maximum when the variable level is equal to or higher than 20).

Visible-hidden: Allows displaying or hiding the bargraph according to the state of the specified Boolean variable.

Direction of the bargraph evolution

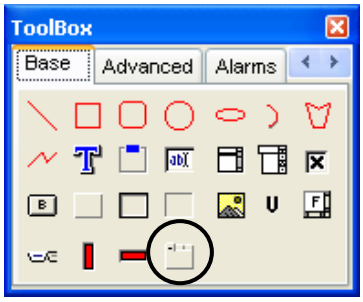
The direction of evolution of the vertical bargraph is the direction of the design layout. If you draw the bargraph from top to bottom, this one will move from top to bottom (100% to 0). If you draw it from bottom to top, this one will move upwards.
The direction of evolution of the horizontal bargraph is the direction of the design layout. If you draw the bargraph of right-hand side towards the left, this one will move from the right-hand side to the left. If you draw it left towards the right, this one will move from left to right.

Programming on bargraph

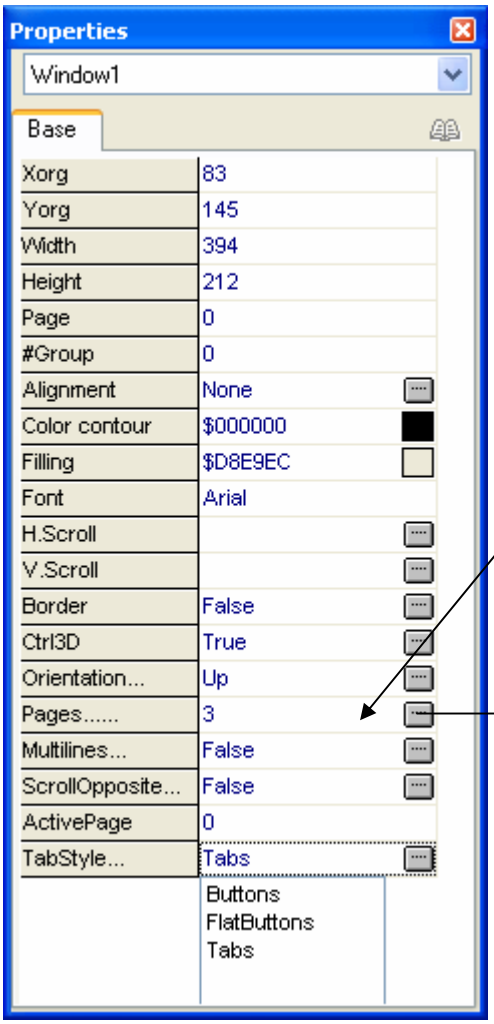
Setting the bargraph value

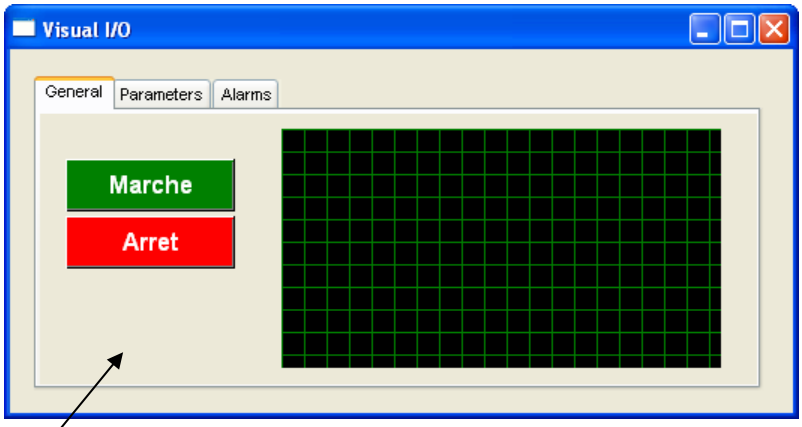
```
// *** BarV1 ************************************
SObject Procedure BarV1;
Begin
  Setvalue(A0); // ** set the bargraph value directly
End;
```

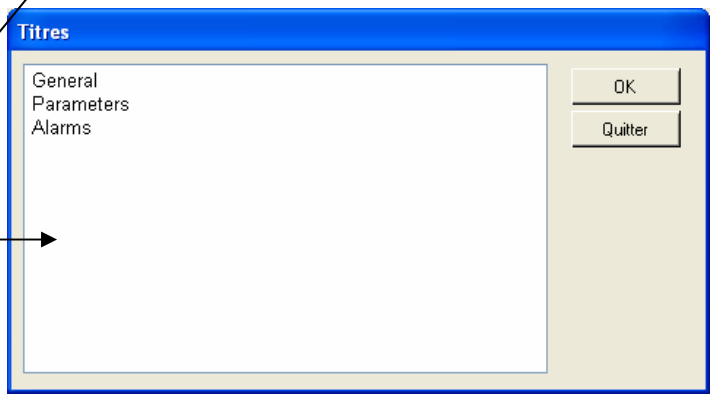
Page Control - Folders



The Page Control component does not have expert. It is enabled to build a multiple page dialog or tabbed notebook type by adding components to each page of the control - you might sometimes want to host an entire form in a page.
These folders can contains all the graphic shapes of Visual I/O. This control allows defining several logical pages or sections of information inside the same window.
Setting number of tabs is done by calling the properties box strike **<F6>**







Double click on the cell 'Pages', a dialog box is displayed to define the headings of each page.

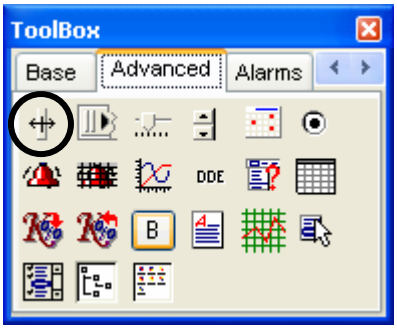
Different tabs are possible in **Tabstyle**. The basic style is Tabs (as shown above). **Orientation** allows the tabs to be displayed above or below.
ScrollOpposite allows when the control page is not broad enough to pass a part of the tabs below.

Programming on page control

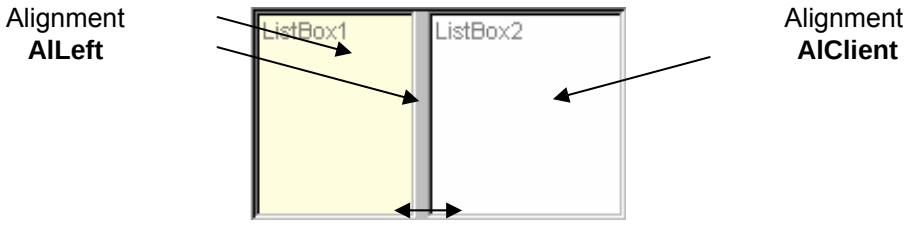
```
// *** Button3 ************************************
SObject Procedure Button3;
Begin
 if Button Then
 Begin
   Go2('Window1');
   Setpage(Getpage+1); // *** go to the next page
 End;
End;
```

☞ **Note**: The first page has the number 0.

Splitter



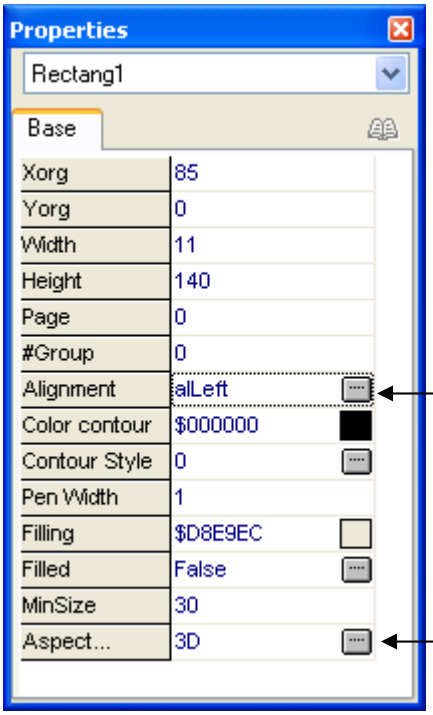
Add a separator into form between 2 aligned controls. To enable the users to resize these controls at execution time, the separator is located between a control aligned on one of the borders of the window and controls filling the remainder of the client zone.
It allows getting your application in good design style. Put one component into the form and set Align property to alLeft. Put Splitter into the form. Put other components and set Align property to AlClient value.
Splitter has MinSize property. If the Align property is alLeft, the splitter can't resize the regions to its left smaller than MinSize pixels..



<u>Properties of the splitter strike **<F6>:**</u>

MinSize to specify a minimum size the splitter must leave when resizing its neighbouring control.

Aspect: Sets the aspect of the splitter 3D or flat.



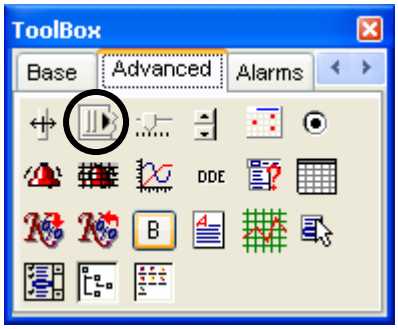
<u>Splitter properties.</u>

The control name is inherited the rectangle with additional properties.

Alignment Alleft: come on to the control from its left (Listbox1) which is already aligned to the left.
Minsize: to specify a minimum size the splitter must leave when resizing its neighbouring control (here ListBox1).

Aspect : 3D or flat aspect.

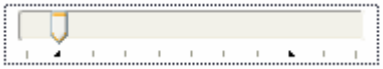
Coolbar



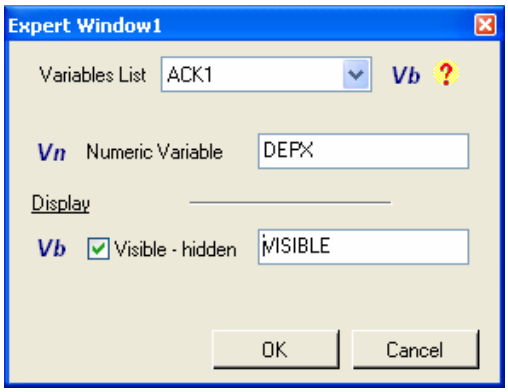
The CoolBar component is a new type of ToolBar. A CoolBar component is a container control that typically contains two or more CoolBands (Windows) which may be resized and rearranged by the user at run time. A CoolBand is a region of the control that can contain other window controls, edit boxes, combo boxes and animations.

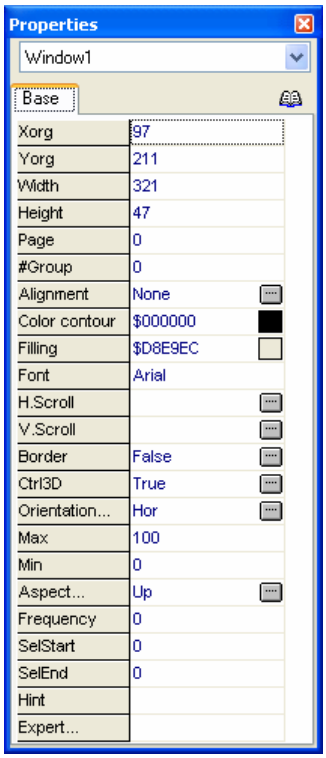
Each bands of CoolBar receive only Windows controls. To add a Windows control into a band, you can choose it in the toolbox of Visual I/O, and then trace it in the desired bar.
Windows Control automatically adjusts with the width and the height of the band (Case above of Edit1)
If you wish to place several controls in a band, trace before a new window (Window of the toolbox) this one auto adjust with the size of the band.
Color to its gray color in contour and in filling (properties box in Visual I/O) and set its H-Scroll properties and V-Scroll with false.
You can trace any element managed by the band in this window.
To add a new Band: Increase the general framework of the CoolBar, and then bring a new Window control into the increased zone. A new bar takes shape automatically. **AutoSize=True** of the property box allows recutting the foot of CoolBar correctly while aligning on the bottom of the last band automatically.

Trackbar



This is a linear gauge displaying a pointer that indicates some value within a specified range, and allowing users to drag the pointer to other values within the range.
Visible-hidden: Allows displaying or hiding the Trackbar according to the state of the Boolean variable





The cells Max and Min to set the Minimum and maximum values which will be assigned to the numerical variable specified in the expert dialog box.
Aspect : allows changing the shape of the cursor.
Frequency : Tick marks are displayed at some specified frequency along the trackbar.
Orientation : allows determining whether the trackbar is on horizontal or vertical position.

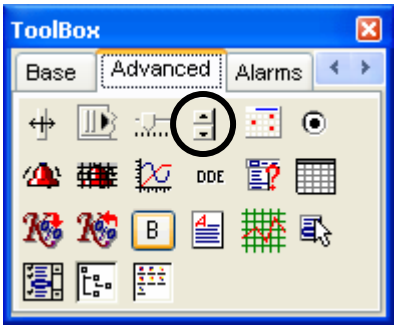
Programming on trackbars

GetTrackBar and **SetTrackbar** set and return the current value of the trackbar.

The trackbar also makes it possible to show two marks: a minimum mark and a maximum mark below the trackbar added to the current graduations. Use procedures **SetTrackStart**(value); **SetTrackEnd**(Value);

There are 2 procedures which do not limit the trackbar. This must be made by program (For example, comparison with max and limitation of position by SetTrackBar (MaxValue)).

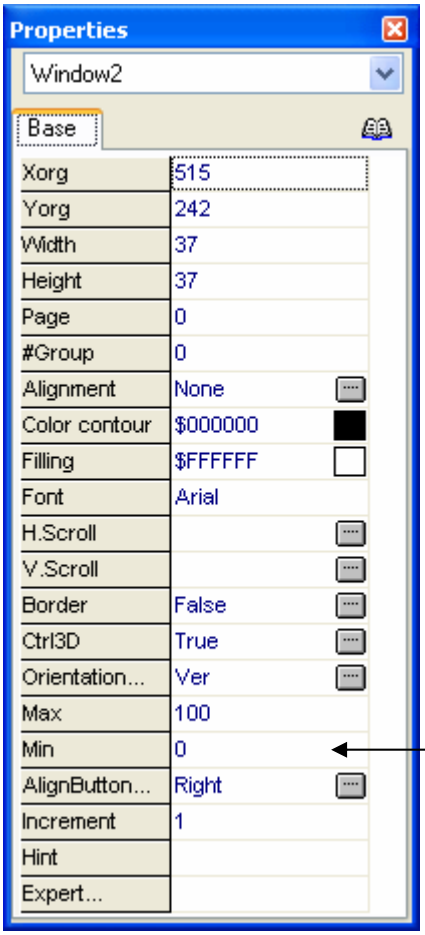
Updown



Up-down controls consist of a pair of arrow buttons, such as the arrows that appear in a spin box. Up-down controls allow users to change the size of a numerical value by clicking on arrow buttons.
Case with editbox:
The up-down control automatically positions itself on the outer edge of the companion control and adjusts its height to match the companion control.

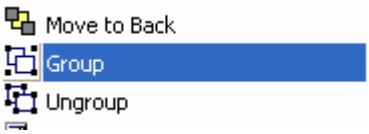
Case grouped with an EditBox

When the user clicks on the editbox or the buttons, focusing is allowed to the text of control. The user can enter a value directly or use the buttons to increase or decrease the value.



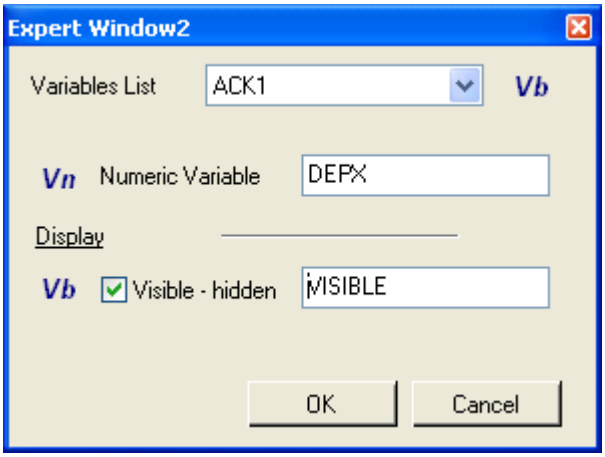
Increment specifies the amount. The position value changes each time the up or down button is pressed.

For grouping you can select 2 objects, and then apply the group command into the popup menu (with right click of the mouse).



AlignButton places the UpDown control on the right or the left of an editbox.

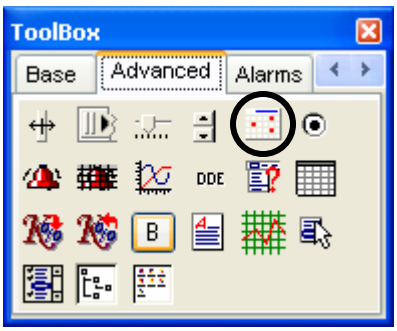
The Maximum and Minimum values, which are set to the numerical variable specified in the expert, are settable in the properties box in cells named **Max and Min**.



The numerical value of increment is specified in the filed of Numeric Variable.

Visible-hidden: Enables displaying or hiding the up/down control according to the state of the specified Boolean variable

DateTime Picker

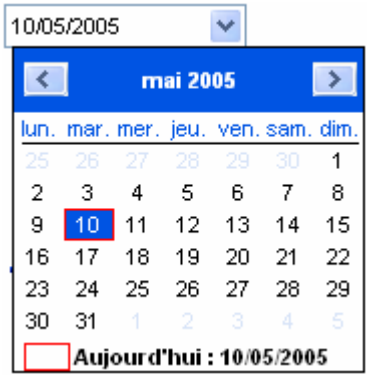


TDateTimePicker displays a list box for entering dates or times.

Users can select a date from the calendar. Dates or times can also be selected by scrolling with Up and Down arrows and by typing.

The underlying Windows control is known to behave unpredictably when used to set dates in 1752, the year English-speaking countries changed to the Gregorian calendar.

At runtime



Programming on DateTimePicker

Get the selected date

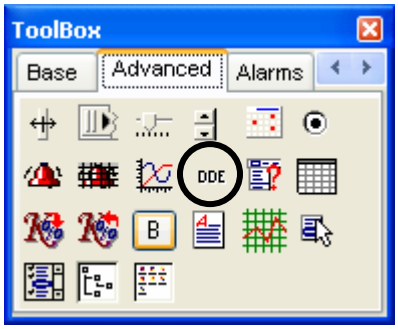
```
// *** Window5 ***************************************
SObject Procedure Window5;
Begin
  CurDate:=GetText; // ** get the date or the time displayed
End;
```

Add 3 days to the selected date

```
// *** Window5 ***************************************
SObject Procedure Window5;
Var
 R,R1 : Real;
Begin
  LaDate:=GetText;            // ** get the date displayed
  R :=StrtoDateTime(Ladate);  // ** Conversion
  R1:=StrToTime('12:00:00');  // *** 12 hours
  R1:=(R1*2)*3;               // ** 24 Hours * 3 days
  R:=R+R1;                    // ** Add 3 days to the original date
  LaDateBis:=DateTimeToStr(R);      // *** Date + 3 days in string format
End ;
```

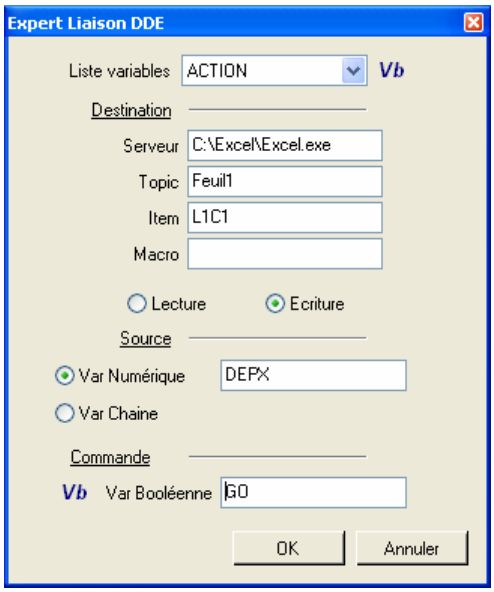
StrToDateTime : Converts a string to a TDateTime value.
StrToTime : Converts a string to a TDateTime value (time only).
DateTimeToStr : Converts a TDateTime value to a string.

DDE Client



The DDE connexion send or receive variable in an DDE server application (ex: Excel).
The DDE enables an application to act as a DDE client. The server application must be launched. One of the applications is a server while the others are clients. The server can also be the client of other applications.
Mechanism is obsolete today ! We just keep it here for compatibility.

☞ **Note** in the case of a communication with Excel, you can use the functions included in the **EXCEL.VPU** library. An example is delivered with the product See project : *Excel interface*



Server DDE server Name.
Topic Conversation
Item Conversation Item

Macro Macro to be sent to DDE server.

Read or Write in the server.

Source :
Numerical Var Numeric variable to be sent to the server
String Var String variable to be sent to the server.
Command:
Boolean Var: Boolean Var validates conversation DDE towards the server.

☞ **Note:** By showing the complete way of the DDE Server, Visual I/O opens this server on the first communication if this application is not already launched.

However, if the DDE Server application is not launched, the first communication is lost.

If you wish to run a communication with several elements (Items), you can use the instructions:

OpenDDE, CloseDDE, DDEGetData, DDESendData.

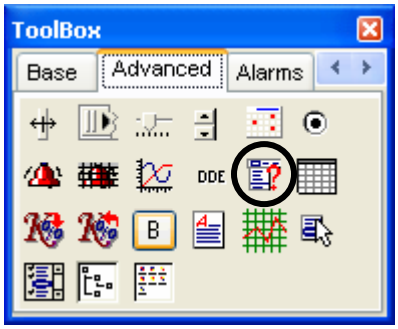
Programming with the DDE mechanism

Fill the 10 first column in an Excel sheet

```
// *** Bmp2 ************************************
SObject Procedure Bmp2;
Begin
  IF Go Then
  Begin
    OpenDDE('C:\Excel\Excel.exe'); // ** Open Excel
    For I :=1 To 10 do
       DDESendData('Feuil1','L1C'+IntToStr(I),IntToStr(I)); //* send to excel
    CloseDDE;
  End;
End;
```

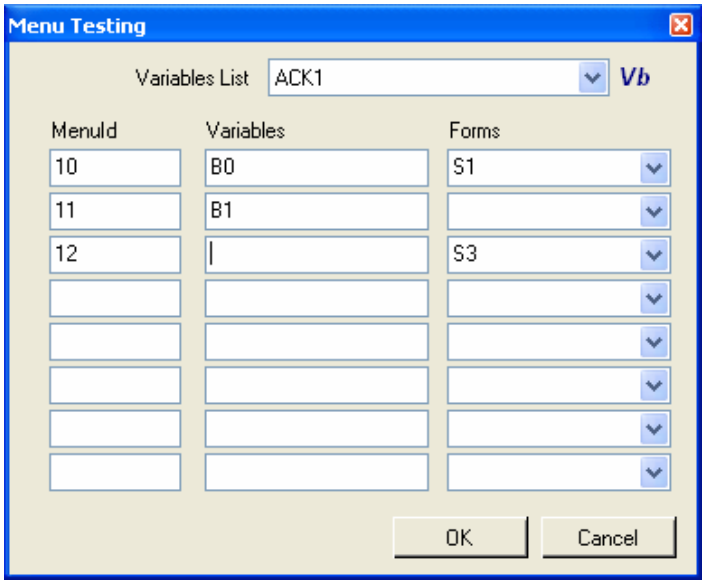
☞ **Note**: A **DDE** server for Visual I/O and Visual PLC applications is available freely on the ARSOFT's Web site (www.arsoft-int.com).

Test Main Menu



This expert compares the menu identifiers with the selected identifier.

It allows setting a variable or calling a form.



The **MenuId** column indicates the identifiers of heading menu.
The **Variables** column indicates the variables to be set when the user clicks on the heading according to the specified identifier in the MenuId column.

In the column 'Forms', you can indicate the name of form to display when the user clicks on the heading who have the specified identifier in the 'MenuId' column.

☞ **Note**: See the chapter of generating a Main Menu. Instead of testing the texts of the menu, Visual I/O assigns a single number (Id) to each menu. This facilitates the test, knowing that number ID is stable and the programmer can change the menus' text without modifying the program.
Also the programmer can run with popupmenu identifiers.

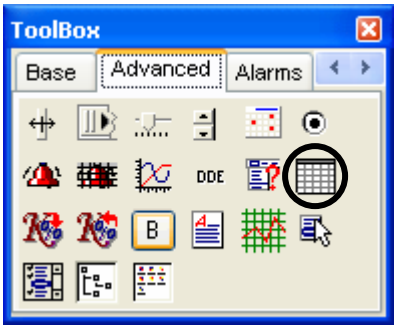
Programming menu testing

Test a click on a menu

```
// *** Button3 ************************************
SObject Procedure Value1;
Begin
 If WinMenu='S1' Then  // ** if the menu clicked is in the form S1
 Case Menuid of
   10 : Display('Parameters');
   11 : B0:=True;
   12 : B0:=False;
   13 : CloseAppli;
 End;
End;
```

☞ **Note** : *The variable **WinMenu** returns the Form name in which the menu was clicked.*
*The variable **MenuStr** returns the item caption clicked in the menu.*

DataGrid

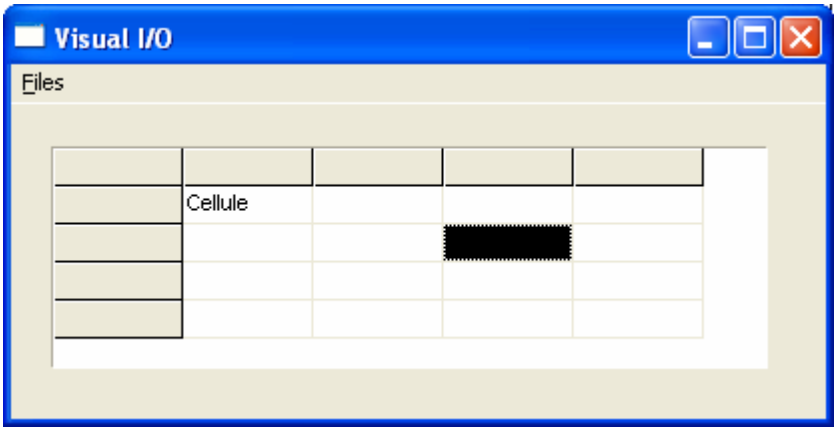


A grid represents informations in a tabular format.

Visual I/O enables many functions to handle by program the data in the grid. A grid can contain text, bitmaps and Windows controls (Listbox, ComboBox, checkbox, palette), lines in different colors.

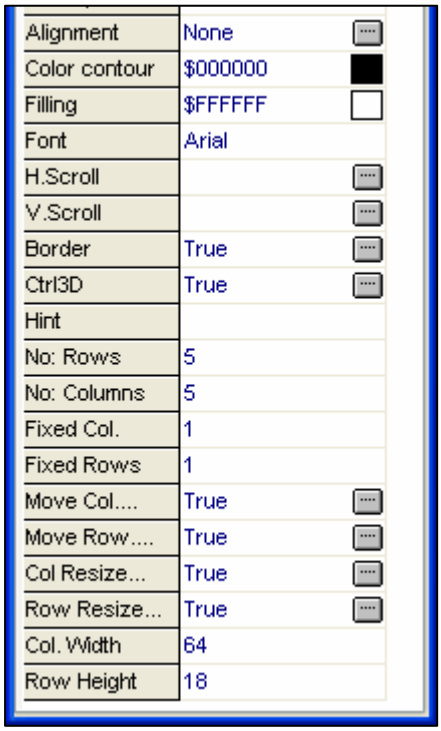
This grid with 5 fixed lines and 5 fixed columns, is symbolized by grey cells in which no editing is possible.

Nonscrolling rows appear at the top of the grid, and are always visible, even when the user scrolls the other rows in the grid. You can use nonscrolling rows for displaying column titles or column numbers.



The grids make it possible that Visual I/0 shows or edits texts in cells in an application. The cells are organized in Lines and Columns. The first column and the first line are numbered 0.
More than 1 million **Cells** can be created.

<u>Hit **<F6>** to display the properties box:</u>



Alignment	None
Color contour	$000000
Filling	$FFFFFF
Font	Arial
H.Scroll	
V.Scroll	
Border	True
Ctrl3D	True
Hint	
No: Rows	5
No: Columns	5
Fixed Col.	1
Fixed Rows	1
Move Col....	True
Move Row....	True
Col Resize...	True
Row Resize...	True
Col. Width	64
Row Height	18

Border and Ctrl3D to modify the grid contour.

No Rows : Specifies the number of rows in the grid..
No Columns : Specifies the number of columns in the grid.

Fixed Col. : Specifies the number of columns on the left of the grid that cannot be scrolled.
Fixed Rows : Specifies the number of rows on the top of the grid that cannot be scrolled.

Move Col.: Scrollable columns can be moved by using the mouse to click on a fixed column and move it on another one (inverse 2 columns).
Move Row: Scrollable rows can be moved by using the mouse to click on a fixed row and move it on another one (inverse 2 rows).
Col Resize: Scrollable columns can be individually resized.
Row Resize: Scrollable rows can be individually resized

Row Height: Height of the rows at initialization time.

Programming on grids

Display a text in red in fixed rows

```
// *** Window1 ************************************
SObject Procedure Window1;
Begin
  If go Then
  Begin
   GSetTextFixedColor(255); // Set the color writing in the fixed cells
   GsetText(1,0,'Title1'); // *** writing in Column 1
   GsetText(2,0,'Title2'); // *** writing in Column 2
   GsetText(3,0,'Title3'); // *** writing in Column 3
   GsetText(4,0,'Title4'); // *** writing in Column 4
  End;
End;
```

Saving the contain of a grid in a text file

```
// *** Window1 ************************************
SObject Procedure Window1;
Var
 Fic,X,Y : Integer;
 TS : String;
Begin
  If Go then
  Begin
    FileDelete('Temp.TXT');   // ** delete the final file
    FileTextAppend(Fic,'Temp.TXT'); // ** open the file for adding
    For X:=0 To GGetColCount-1 Do  // ** Scan all the columns
     For Y:=0 to Ggetrowcount-1 do // ** Scan all the rows
     Begin
       TS:=GGetText(X,Y); // ** get the contain of the grid
       Writeln(Fic,TS);   // ** write the string value
     End;
    FileTextClose(Fic); // ** close the final file
  End;
End;
```

Load the contain of a text file in a grid

```
// *** Button2 ************************************
SObject Procedure Button2;
Var
 Fic,X,Y : Integer;
 TS : String;
Begin
  If LoadGrid and (FileExists('Temp.TXT')) then // ** if the file exists
  Begin
    Gcleargrid; // ** Clear all the grid
    FileTextOpen(Fic,'Temp.TXT'); // ** open the file for reading
    For X:=0 To GGetColCount-1 Do  // ** Scan all the columns
     For Y:=0 to GgetRowCount-1 do  // ** Scan all the Rows
     Begin
       Readln(Fic,TS);   // *** read a line in the text file
       GSetText(X,Y,TS); // ** write in the specific cell
     End;
    FileTextClose(Fic); // ** close the final file
  End;
End;
```

☞ **Note** : The program which safeguards the contents above is written behind the concerned grid. The program to load the contents of a file is written behind the button. It is possible if the form includes one grid. Then all instructions concerning the grid are sent to the unique grid. In the case of several grids, it is necessary to execute a Go2 procedure before sending to specify the destination grid.

Detect if a change occurs in one of the cell of the grid and colorize it in red

```
// *** Window1 ************************************
SObject Procedure Window1;
Var
 X,Y : Integer;
Begin
  If Gcellchange Then  // ** detect if a change occurs
  Begin
    X:=Ggetcolchanged;  // ** Gets the modified column
    Y:=Ggetrowchanged;  // ** Gets the modified row
    Gsetcellcolor(X,Y,ClRed); // ** colorize the cell background
  End;
End;
```

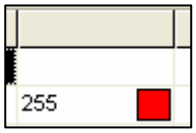
Enabe disable the editing in the cell

```
// *** Window1 ************************************
SObject Procedure Window1;
Var
 X,Y : Integer;
Begin
  GGetcurrent(X,Y); // ** Get the cursor position in the grid
  If (X=1) and (Y=2) Then Gdevalidedit // ** Disable if the cursor is in Col1 and
                     Else Gvalidedit;  // Row 2 and Enable the editing in
                                       // the others cells
End;
```

Accept the drag & drop coming from an editbox

```
// *** Edit1 **************************************
SObject Procedure Edit1;
Begin
  Drag(Gettext); // ** the drag & drop comes from Edit1
End;
// *** Window1 ************************************
SObject Procedure Window1;
Var
 X,Y : IntegeTS : String;
Begin
  if Drop Then // ** Drag release on the grid
  Begin
    Flyingcells(X,Y); // ** returns X/Y of the cell below the mouse
    Gsettext(X,Y,Getdrop);
  End;
End;
```

Call the palette color from a cell

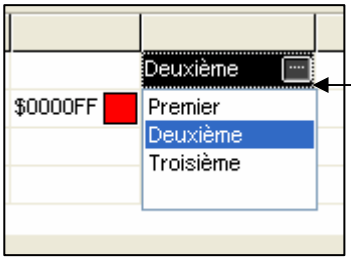



```
// *** Window1 ************************************
SObject Procedure Window1;
Begin
  if Start_visu then
  Begin
    GCommand('Color',2,2);      // *** the cell is a cell color
    GSetText(2,2,'$0000FF');     // *** paint the cell in red ***
    GSetControl(2,2,'Palette'); // *** the cell allowed by double-click
                                // to call the palette of color
  end;
End;
```

Display a listbox in a cell

It is possible to display Windows controls (ComboBox, ListBox and Checkbox) in the cells.
These controls must be created in the form by the graphic editor with their various possible types (DropDown/ Simple/ ownerdraw/ sorted etc.).
A program will manage the contents and the choice of the operator in the list before assigning it to the cell.
The graphic editor proposes a name (Combo1, Switch2 etc.). This name will be used to assign this control to a particular cell.
Control must be hidden at the launching of the application using the **HideControl** procedure. The grid will display it in the concerned cell. The object (listbox) must control the click mouse, the recovery of the character string and the assignment of this one to the cell.

```
// Creating a program in the listbox which must appear in the cell of the grid
// *** ListBox1 ************************************
SObject Procedure ListBox1;
Begin
 If Start_Visu Then     //** At the startup of the application
 Begin
    Addstring('Premier');   // Add a text 'Premier' into the listbox
    Addstring('Deuxième');  // Add a text 'Deuxième' into the listbox
    Addstring('Troisième'); // Add a text 'Troisieme' into the listbox
    HideControl;            // Hide the listbox
  End;
// If a click is done in the listbox, the text is written in the current cell
// of the grid.
  If  TopLBdown then
    GSetTextCurrent(GetSelString);  //Get the string and send it into the current
cell

// *** Window1 *************************************
SObject Procedure Window1;
Begin
  if Start_visu then
  Begin
    GCommand('Color',2,2);        // *** the cell is a cell color
    GSetText(2,2,'$0000FF');      // *** paint the cell in red ***
    GSetControl(2,2,'Palette');  // *** The cell allows calling
                                 // *** the palette color by double click
    GCommand('CONTROL',3,1);     // ** Column 3 and Row 1 contain a control
    GSetControl(3,1,'Listbox1'); // ** the control is the listbox1
  End;
End;
```

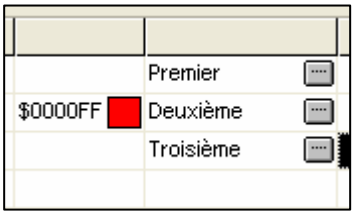


When a cell is affected by a control, a small button on the right of it appears to display a choice.

When double clicking on the cell (3,1), the listbox appears under the cell which displayed its lines. The program in the listbox1 is active. When the operator clicks on it, it writes the text in the cell, and then the grid hides this Listbox automatically.

Display the same listbox in several cells

```
// ** the same listbox can be used as many time as desired.
// *** Window1 ****************
SObject Procedure Window1;
Begin
  if Start_visu then
  Begin
    GCommand('Color',2,2);        // *** the cell is a cell color
    GSetText(2,2,'$0000FF');      // *** paint the cell in red
    GSetControl(2,2,'Palette');   // the Cell by double-click to call the
                                  // palette of colors
    GCommand('CONTROL',3,1);      // ** column 3 and row 1 include a control
    GSetControl(3,1,'Listbox1'); // ** The control is the listbox1
    GCommand('CONTROL',3,2);      // ** column 3 and row 2 include a control
    GSetControl(3,2,'Listbox1'); // ** The control is the listbox1
    GCommand('CONTROL',3,3);      // ** column 3 and row 3 include a control
    GSetControl(3,3,'Listbox1'); // ** The control is the listbox1
  End;
End;
```



Centring texts in columns

The **GSetOptions** procedure is allowed to fix the options of display of the grid.

Various options are possible by a properties box or by passage of parameters in the procedure GSetoptions with the combinations of constants which are declared in Windows.pas.

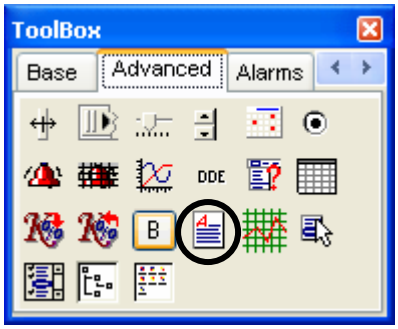
```
Const
 AlignCenter   = 16384;
 AlignRight    = 32768;

GridOptions1 = FixedHorzLine + FixedVertLine + HorzLine + VertLine +
                DrawFocusSelected + RowSizing + ColSizing + RowMoving +
                ColMoving;

GridOptions2 = FixedHorzLine + FixedVertLine + HorzLine + VertLine +
                  RowSizing + ColSizing + RowMoving + ColMoving;

GridOptions3 = FixedHorzLine + FixedVertLine + HorzLine + VertLine +
                  RowSizing + ColSizing;
```

Example: Modify the attributes behind the grid of data. Here texts will be centred in columns.

```
Uses Windows; //*** For using predefined onstants
{*** Window1 ****************************************}
SObject Procedure Window1;
Begin
  If Start_Visu Then  //*** Once at launch time of the application
   Gsetoptions(GridOptions1 + AlignCenter );
End;
```

☞ **Note** For additional information on GSetoptions, see instructions system on data grids.

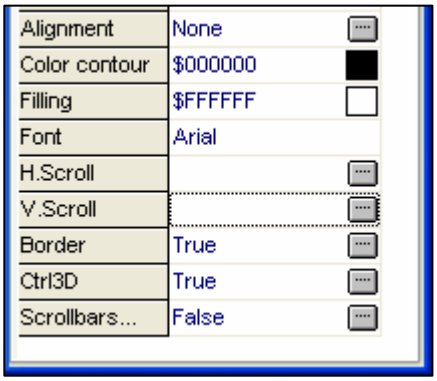
Richedit - Rich Text Edit Control



Use a RichEdit object to put a standard Windows rich text edit control on a form. Rich text edit controls let the user enter text that includes variation in font attributes and paragraph formatting information.

The procedures and functions to handle a Richedit are contained in the library RICHINT.VPU whose source is provided with Visual I/O, report to the chapter library RICHINT.

Properties Box of Richedit is restricted and resembled to an internal window. ►



The property **Scrollbars** determine whether it is necessary to add scrollbars to the control when the text is too large or too long for a complete display.
The properties Trace color, Font, H.Scroll, V.Scroll are inactives.

The Richint library

This library is described in detail in the chapter Richint library. It allows using procedures and functions to control a Richedit control. Few procedures are from the library:

Function RichAddstring *(Value : String; Handle : Integer) : Integer;*
Procedure RichClear *(Handle : Integer);*
Procedure RichSetText *(PTText : Integer; Handle : Integer);*
Function RichGetCount *(Handle : Integer) : Integer;*
Function RichGetTextlen *(Handle : Integer) : Integer;*
Procedure RichInsert *(Index: Integer; S: string; Handle : Integer);*
Procedure RichLoadFromFile *(Filename : String; Handle : Integer);*
Procedure RichSaveToFile *(Filename : String; Handle : Integer);*
Procedure RichClearSelection *(Handle : Integer);*
Procedure RichCopyToClipboard *(Handle : Integer);*
Procedure RichCutToClipboard *(Handle : Integer);*
Procedure RichPasteFromClipboard*(Handle : Integer);*
Procedure RichUndo *(Handle : Integer);*
Procedure RichSelectAll *(Handle : Integer);*
Procedure RichSetProtected *(Value: Boolean; Handle : Integer);*
Procedure RichSetColor *(Value: TColor; Handle : Integer);*
Procedure RichSetFontSize *(Value: Integer; Handle : Integer);*
Function RichSetFontname *(Value: String; Handle : Integer) : Boolean;*

☞ **Note** : See the chapter for details of these procedures.

Programming on richedit control

Loading a file in a rechedit control

Saving the containing of a richedit control in a file

```
Uses Richint;
// *** Window1 ************************************
SObject Procedure Window1;
Begin
  // ** load the file into a richedit
  If Openf  Then RichLoadFromFile ('C:\Description.Rtf');
  // ** Save the Richedit in a file
  If Savef  Then RichSaveToFile   ('C:\Descriptionx.Rtf');
  If ClearF Then RichClear(Gethwnd); // ** clear the richedit
  OpenF:=False;
  Savef:=False;
  ClearF:=False;
End;
```

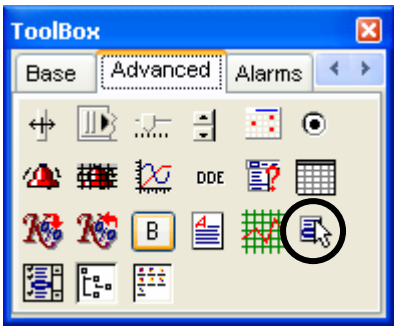
☞ **Note**: if the procedure **RichLoadFromFile** and **RichSaveToFile** is used behind the richedit component, only one parameter can be passed because the procedure works on the current richedit control.
Thus the file must be loaded into another richedit. You need to pass a supplementary parameter which is the handle of the concerned richedit.

The optional parameters on the procedures are available from the version 7.00 of Visual I/O.

Load a file in another richedit

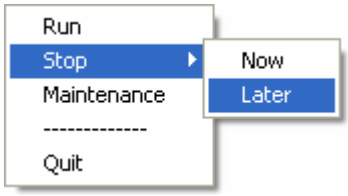
```
Var
  HDR : Integer ;
// ** get the handle of the richedit at launching time of the application
// *** Window2 ************************************
SObject Procedure Window2;
Begin
  If Start_visu then HDR:=GetHwnd;
End;

// *** Button1 ************************************
SObject Procedure Button1;
Begin
  // ** load the file into the richedit its handle is HDR
  // ** HDR target handle
  If Openf  Then RichLoadFromFile ('C:\Description.Rtf',HDR);
End;
```

Popup Menu



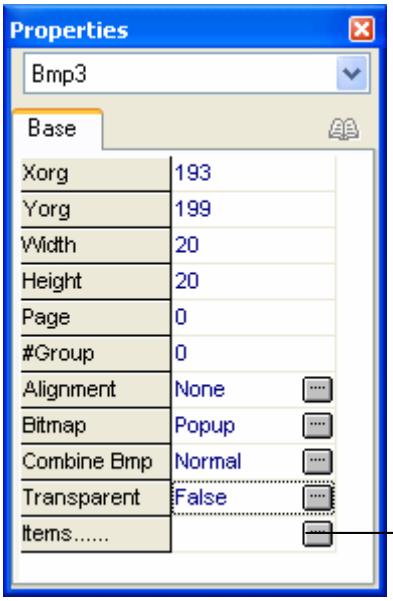
To define the pop-up menu which appears when the user clicks on a control with the right mouse button.

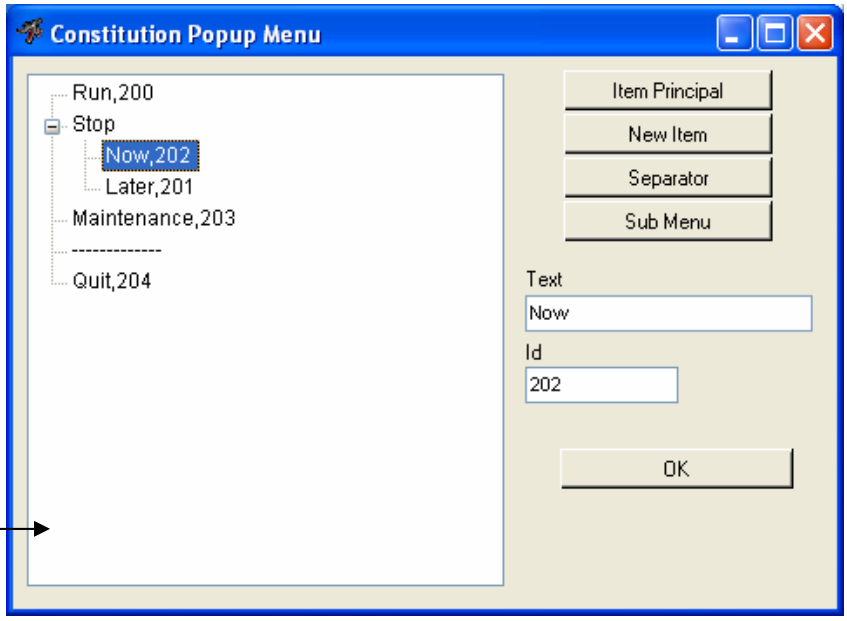
Example of a pop-up menu



Create a pop-up menu

Place the component on the form, and then hit <F6> to call the properties box. Only the parameter items are used. Double-click on it to call the Pop-menu conception.



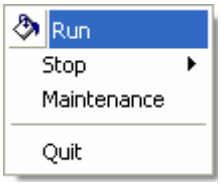


The principle of creating the items of a pop-up menu is similar to the main menu.
Identifiers are associated with the texts of the pop-up menu allowing a test by program. Even these texts change according to the used language. Identifiers increase it automatically, but begin at 200 for the first item. It is possible to modify this identifier in the editbox ID.

Programming on pop-up menus

Enable or disable a menu item

```
// *** Bmp2 ************************************
SObject Procedure Bmp2;
Begin
 If Go Then ValidMenuItem(201,True)
       Else ValidMenuItem(201,False);
End;
```

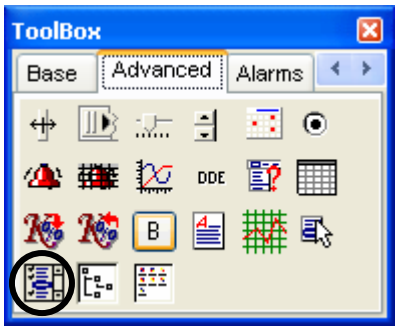


Assign a bitmap to a menu item

```
// *** Window1 **********************************
SObject Procedure Bmp2;
Begin
  If Start_Visu Then SetMenuBitmap(201,'menupalette');
End;
```

☞ **Note** : The bitmap must be in 16 colors with the transparent color given by the pixel on bottom/left of the bitmap. This bitmap can be included in the final executable or be present in the current directory.

CheckListBox



This component displays a scrollable list with check boxes next to each item.

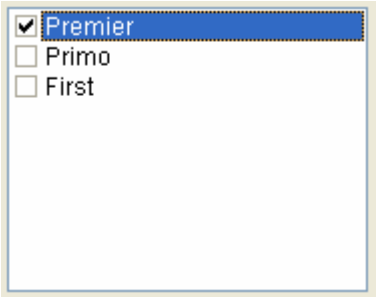
CheckListBox is similar to ListBox, except that each item has a check box next to it. Users can check or uncheck items in the list.

Differences with classical listboxes

The component CheckListBox is a simplified version of the listbox.
This control allows the current operations of a listbox: Add, insertion, Clear some lines, the possibility of check/uncheck checkboxes and testing theirs states.

Programming on checklistbox

Check or uncheck an item in the list




```
// *** ListBox1 ************
SObject Procedure ListBox1;
Begin
  If Start_visu Then
  Begin
    Addstring('Premier'); // add lines to the listbox
    Addstring('Primo');
    Addstring('First');
    CheckListSetState(0,True); // ** check the first
    CheckListSetState(1,False); // item
    CheckListSetState(1,False); // ** Uncheck the second item
  end;
End;
```

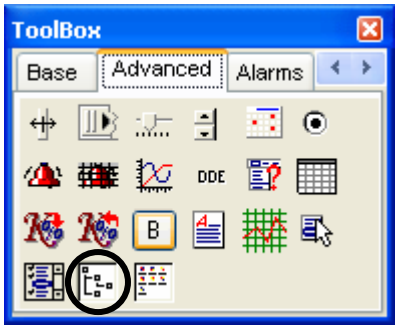
Test the checkbox of a line in a listbox

```
// *** ListBox1 ************************************
SObject Procedure ListBox1;
Begin
   Go:=CheckListGetState(0); // ** Go=True if the item is checked
End;
```

Test if an item is checked in the listbox

```
// *** ListBox1 ************************************
SObject Procedure ListBox1;
Begin
  Go:=False;
  For I:=0 to GetCount-1 do
  Begin
    If CheckListGetState(I) Then Go:=True; // ** test if the item is checked
  End;
End;
```

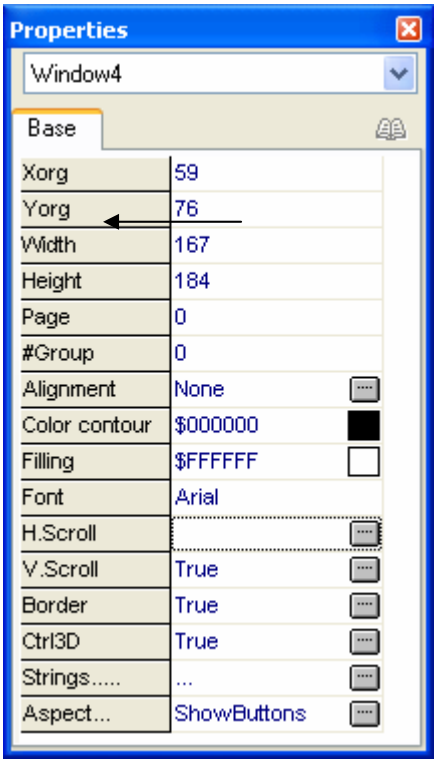
TreeView

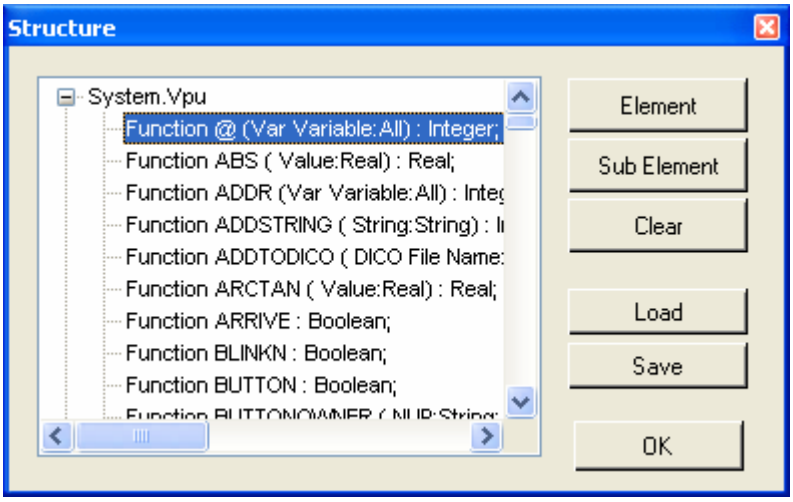


TreeView represents a window that displays a hierarchical list of items, such as the headings in a document, the entries in an index, or the files and directories on a disk.

Description

You can use TreeView to add an expanding and contracting outline to a form. Each node in a tree view control consists of a label and a number of optional bitmapped images. Each node can have a list of subnodes associated with it. By clicking on a node, the user can expand or collapse the associated list of subnodes.

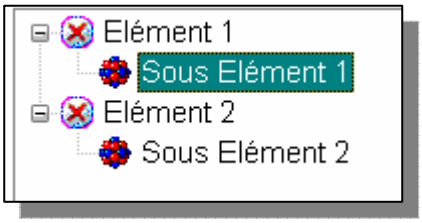




The TreeView do not have expert which you must write program code to use.
Double-click on Strings.., to create the structure and nodes including sub nodes or not.
This dialogbox also allows initializing the treeview with a text file at design time.
Load from a file or save the treeview to a file.

The Aspect property specifies whether to display plus (+) and minus (-) buttons to the left side of each parent item to expand or collapse the child items as an alternative to double clicking on the parent item.
You can modify an item by double-clicking and put in a new value.

Programming on treeview

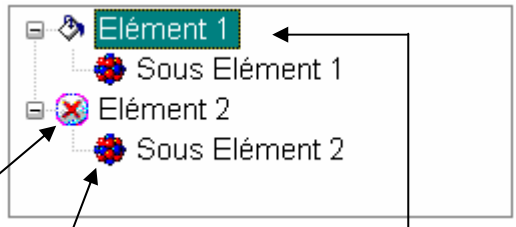


Add images into a treeView

***Attention*: images must be in the same size !!**

```
// *** ListBox1
SObject Procedure Window1;
Begin
  If Start_visu Then
  Begin
    Xaddimage('Closer16'); // ** Bitmap on all nodes 0
    Xaddimage('ATOMES');   // ** Bitmap on all sub nodes 1
  End;
End;
```

Display an different image to treeView item




```
// *** ListBox1 *****************
SObject Procedure Window1;
Begin
  If Start_visu Then
  Begin
    Xaddimage('Closer16'); // ** First Image Level 0
    Xaddimage('ATOMES');   // ** 2nd Image Level 1
    Xaddimage('MenuPalette'); // ** 3rd Image Level 2 missing here
    Xsetimage(1,2);           // ** Force First element image 2
  End;
End;
```

☞ **Note**: The elements are numbered from 1 (higher) to the lower element.

To known the selected item and the number of items in a treeview

```
// *** ListBox1 **************
SObject Procedure Window1;
Begin
 Selection:=TreeGetSelectedItem; // ** Return the number of the selected line
 Count:= TreeGetCount;  // ** Return the number of lines(items) in the treeview
End;
```

Modify the text of a line

```
// *** ListBox1 *****************
SObject Procedure Window1;
Begin
  If TreeGetItem(1)<>'ARSOFT' Then //** Read the first line and compare the text
     TreeSetItem(1,'ARSOFT'); // ** Set a new text on the line #1
End;
```

Searching a text in a TreeView and force the selection

```
// *** ListBox1 ************************************
SObject Procedure Window1;
Begin
  For i:=0 to TreeGetCount-1 Do // ** Scan all the lines
  Begin
    If TreeGetItem(I)='ARSOFT' Then // ** Compare the line and text "ARSOFT"
        TreeSetSelectedItem(I);     // If True, force the selection on the line
  End;
End;
```

☞ **Note**: Forcing a selection displays the corresponding line (expand treeview if necessary)

Set an image to a node only when it is selected

```
SObject Procedure Window1;
Begin
  If Start_visu Then
  Begin
    Xaddimage('Closer16');     // ** First Image level 0
    Xaddimage('ATOMES');      // ** Second Image  level 1
    Xaddimage('MenuPalette'); // ** Third Image Level 2 missing here
    Xsetimage(1,2);           // ** Force First element image 2
    TreeSetSelectedImage(2,2); // Image #2 is displayed when node 2 is selected
  End;
End;
```

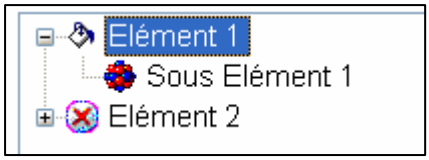
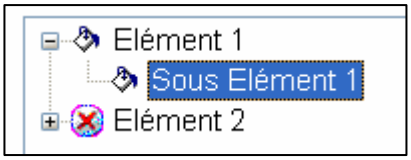
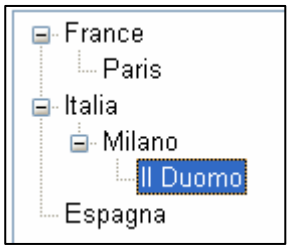


Image #2 is displayed
when node 2 is selected



Clear then add lines in a treeview




```
SObject Procedure Window1;
Begin
  if Start_visu then
  Begin
    TreeLoadFromFile('XXXX');    // Clear the treeview when file is missing
    TreeAdd(0,'France');
    Treeaddchild(1,'Paris');     // Add a sub node to line 1
    TreeAdd(0,'Italia');         // Add a new node to the end(0)
    Treeaddchild(3,'Milano');    // Add a sub node at the line 3
    Treeaddchild(4,'Il Duomo'); // Add a sub node to the line 4
    TreeAdd(0,'Espagna');        // Add a node to end
  End;
End;
```

Sort a treeview

```
// Sorting or resorting the procedure XAlphaSort triggers node
    ...
    Treeaddchild(4,'Il Duomo');
    TreeAdd(0,'Espagna');
    XAlphaSort ;
End;
```

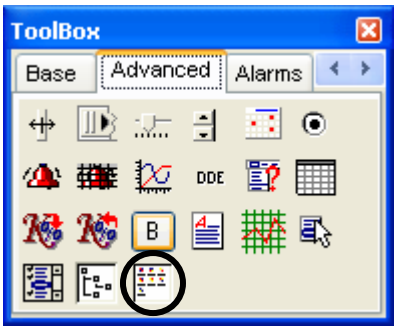
Add lines into a treeview using a file

```
// *** ListBox1 ******
SObject Procedure Window1;
Var
  Fic : Integer;
Begin
  if Start_visu then
  Begin
    FileDelete('Temp');
    Filetextappend(Fic,'TEMP'); // ** Creating an empty file in adding mode
    Writeln(Fic,'France');
    Writeln(Fic,Chr(9)+'Paris');
    Writeln(Fic,'Italia');
    Writeln(Fic,Chr(9)+'Milano');
    Writeln(Fic,Chr(9)+Chr(9)+'Il Duomo');
    Writeln(Fic,'Espagna');
    Filetextclose(Fic);
    TreeLoadFromFile('TEMP'); //  load the File
    XAlphaSort;
  End;
  If Go Then TreeSaveToFile('MODELE.TXT'); // save the treeview in a file
End;
```

Close a treeview

```
SObject Procedure Window1;
Var
  Fic : Integer;
Begin
  If Close Then TreeFullCollapse;
End;
```

Listview

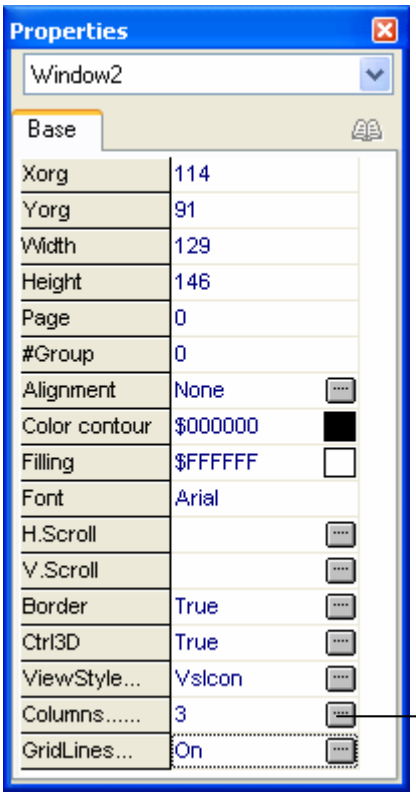


ListView displays a list of items in various ways.
The items can be displayed in columns with column headers and sub-items vertically or horizontally with small or large icons.

Description

The items can be displayed as a set of movable icons, or as columns of text. These are the possible values:

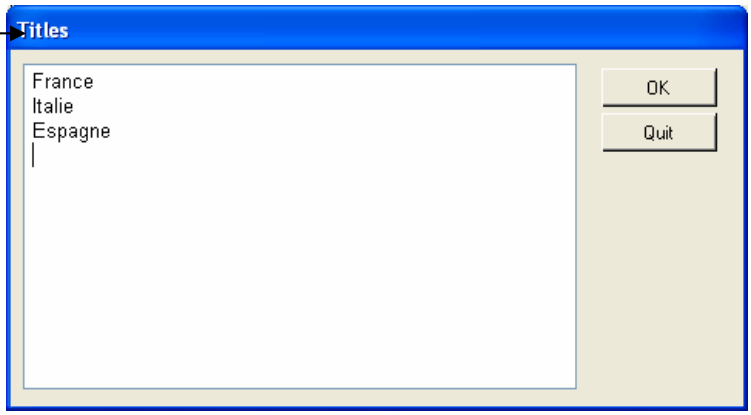
vsIcon Each item appears as a full-sized icon with a label below it. The user can drag the items to any location in the list view window.
vsList Each item appears as a small icon with a label to the right of it.
Items are arranged in columns and cannot be dragged by the user.
vsReport Each item appears on its own line with information arranged in columns. The leftmost column contains the small icon and label, and subsequent columns contain subitems which are specified by the application.



Vsicon

VsList

VsReport

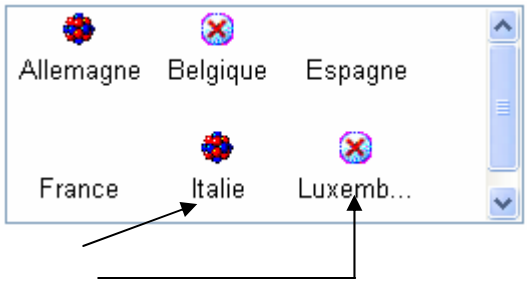


Colums Specifies the text that appears in the 3 possible modes.
Gridlines Determines whether lines are drawn separating items in the list in mode VsReport.

Icons are registering set by program only.

Programming on ListView

Set images behind texts (mode Vsicon and VsList)

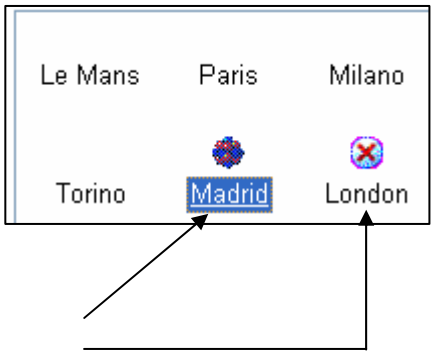



```
// *** Window2 ***************
SObject Procedure Window2;
Begin
 if Start_visu Then
 Begin
   Xaddimage('Atomes');    // Image 0
   Xaddimage('Closer16'); // Image 1
   XSetImage(4,0); // ** assign text 4 image 0
   XSetImage(5,1); // ** assign text 5 image 1
 End;
End;
```

☞ **Note**: With each image recording, those are affected automatically in texts order number

Here image 0 is affected directly by text 0, image 1 by text 1.
Instruction **XSetImage** modifies this effect by setting another image number to the specified text.

Clear and add lines in a listview

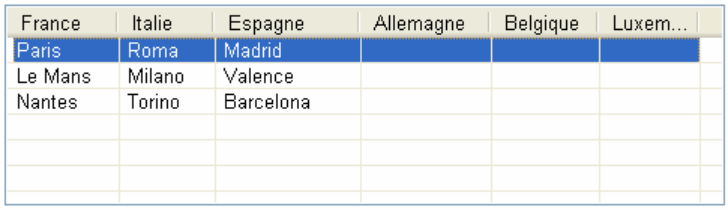



```
// *** Window2 ********
SObject Procedure Window2;
Begin
 if Start_visu Then
 Begin
   Xaddimage('Atomes'); // ** Image 0
   Xaddimage('Closer16'); // ** Image 1
   XSetImage(4,0); // assign text 4 image 0
   XSetImage(5,1); // assign text 5 image 1
     ClearList;    // Clear entire list
     Addstring('Le Mans'); // Add texts
     Addstring('Paris');
     Addstring('Milano');
     Addstring('Torino');
     Addstring('Madrid');
     Addstring('London');
     XSetImage(4,0); // Re Affect to text 4 image 0
     XSetImage(5,1); // Re Affect to text 5 image 1
 End;
End;
```

The preceding images remain recorded but are no more automatically affected. It is necessary to use the **XSetimage** function to rebuild all the necessary assignments.

Add lines in a listview type report

```
SObject Procedure Window2;
Begin
 if Start_visu Then
 Begin  // ** add into column
   Lvadditem(0,1,'Paris');
   Lvadditem(0,2,'Le Mans');
   Lvadditem(0,3,'Nantes');
   Lvadditem(1,1,'Roma');
   Lvadditem(1,2,'Milano');
   Lvadditem(1,3,'Torino');
   Lvadditem(2,1,'Madrid');
   Lvadditem(2,2,'Valence');
   Lvadditem(2,3,'Barcelona');
 End;
End;
```



Each column can be sorted while clicking on its title. The first column (on the left) has Number 0.

☞ **Note** : **Addstring** procedure works but only affects the first column.

Search an element in a listview type report

```
For X:=0 To 2 Do
 For Y:=0 To GetCount Do // ** Line 0 = Title
   Begin
     TS:=LvGetitem(X,Y);
     If TS='Paris' Then
       SetSelIndex(Y); // ** Sets the selection on the line containing 'Paris'
   End;
```

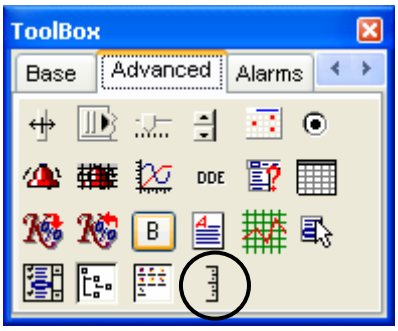
Function and common procedures to the listbox and a listview

```
Procedure   AddString (Value : String);
Function    GetSelString : String;
Function    GetString    ( Index : Integer) : String;
Procedure   DeleteString ( Index : Integer);
Procedure   InsertString ( Index : Integer; Value : String);

Function    GetCount : Integer;
Procedure   ReadOnly ( State : Boolean);
Procedure   ClearList;
```

The following procedures and functions concern the grids. They also work on a report.

```
Function   GGetText(ACol, ARow: Integer): String;
Procedure  GSetText(ACol, ARow: Integer; Value : String);
Function   GGetColCount : Integer;
Function   GGetRowCount : Integer;
Procedure  GSetColWidth (Col : Integer; Width : Integer);
Procedure  GClearGrid;
```

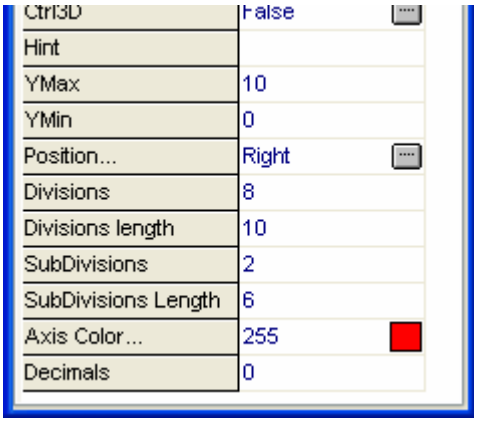
Additional Axis



Use this component to display a supplementary axis by a trend, bargraph, etc.

Description

The properties box is enabled to regulate the different and specific attributes to the axis.



Ctrl3D	False
Hint	
YMax	10
YMin	0
Position...	Right
Divisions	8
Divisions length	10
SubDivisions	2
SubDivisions Length	6
Axis Color...	255
Decimals	0

YMax: Graduation Value (highest) (Maxi).
YMin: Graduation Value (lowest) (Mini).

Position: Right or Left. Graduations is showed on the left or on the right.
Divisions: Number of main divisions.
SubDivisions: Number of sub divisions.

Divisions Length: Length in pixels of each division.
SubDivisions Length: Length in pixels of each subdivision.

Axis Color: Axis divisions and sub-division's color.
Decimals: Number of decimals to show.

Programming on axes

You can dynamically modify the attributes of the axis using the **AxisInt.pas** library (delivered with source code).

AxisYMax : **Real** variable to set the maximum value to draw.

AxisYMin : **Real** variable to set the minimum value to draw.

AxisRight : **Boolean** Variable to draw ticks on the left or the right.

AxisDiv : **Integer** Variable to set the number of major ticks of the axis.

AxisSubDiv : Integer Variable to set the number of minor ticks of the axis.

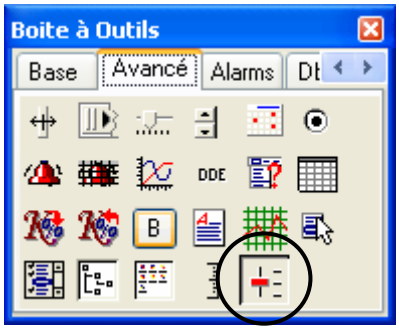
AxisColor : Integer Variable to set the axis color.

AxisDecimals : Integer Variable to set the number of decimal to draw.

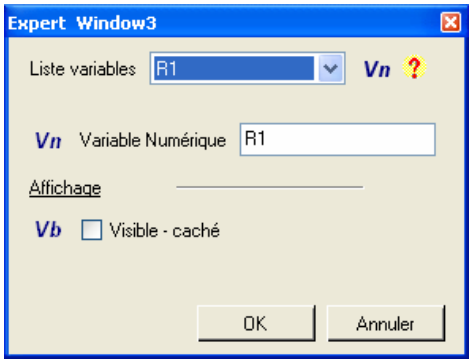
Change the maximum and the minimum of the axis

```
If Change Then
Begin
  Go2('Window1');       //*** go to the axis component
  AxisYMax:=NewYMax;    //*** Affect the new value to the maxi
  AxisYMin:=0;          //*** Affect the new value to the mini
End;
```

Additional Slider



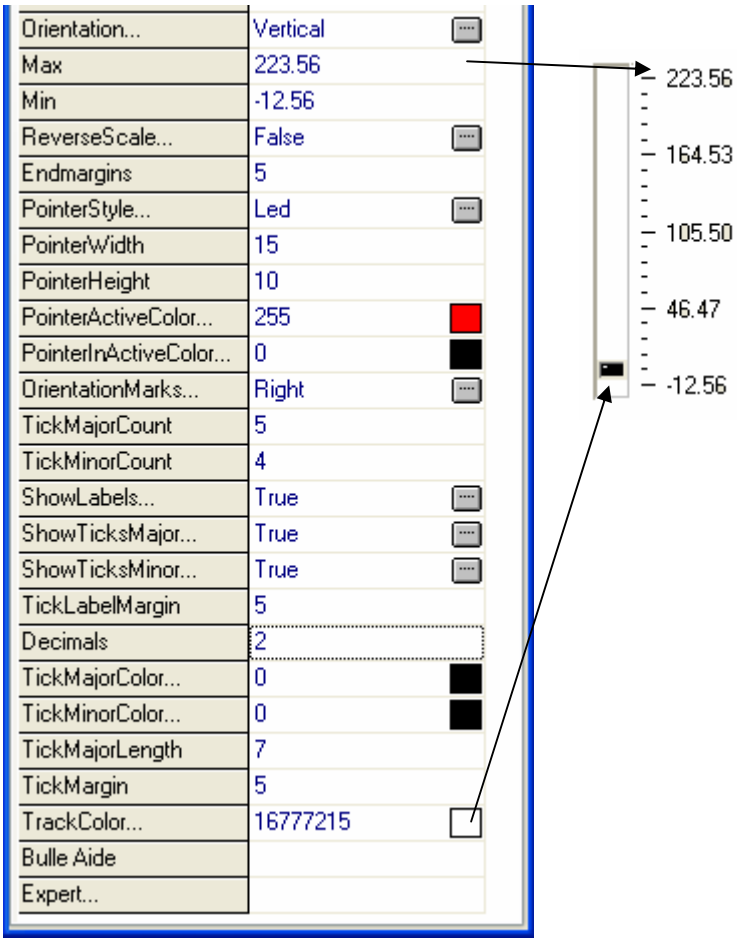
Use this component to set a variable in integer or real type. Thanks to a slider, this component is more complete than windows.
This component enables many aspects.
It is an Analog (Real/Integer) output control component, which supports both keyboard and mouse control.



The expert allow settings for a integer or real variable
This expert is also the value's one.

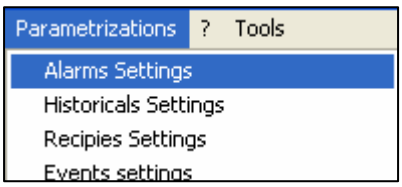
Description

The properties box is enabled to regulate the different and specific attributes to the slider.



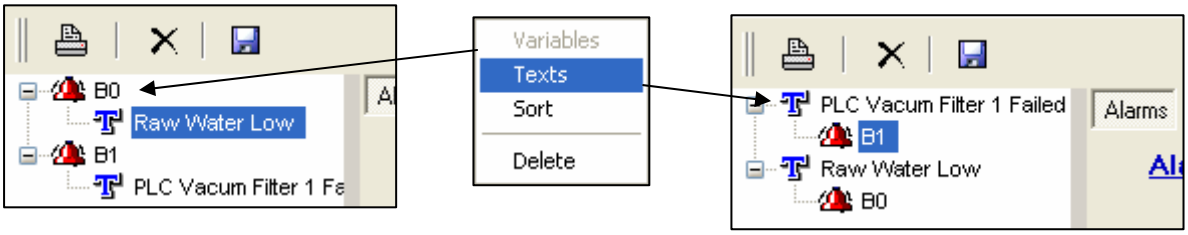
Orientation: vertical or Horizontal.
ReverseScale: Specifies whether the scale is reversed.
EndMargins: Specifies the margin at the ends of the control.
PointerWidth: width of the pointer.
PointerActiveColor: color of the indicator when active. It is active is when the mouse is down.
PointerInActiveColor: color of the indicator when active. It is inactive is when the mouse is up.
OrientionMarks: which side of the control TickMarks are drawn (left or right).
Showlabels, ShowTickMajor, ShowTicksMinor: Use these properties to specify whether Labels and ticks are shown.
TickLabelMargin: Specifies the margin between the Tick Labels and Major Ticks.
Decimals : Decimals drawn at the ticks.
TickMajorColor, TickMinorColor: color of the major and minor ticks.
TickMajorLength: length of the major ticks.
TickMargin: Margin between the Ticks and the Pointer.
TrackColor : color of the track.

Alarms

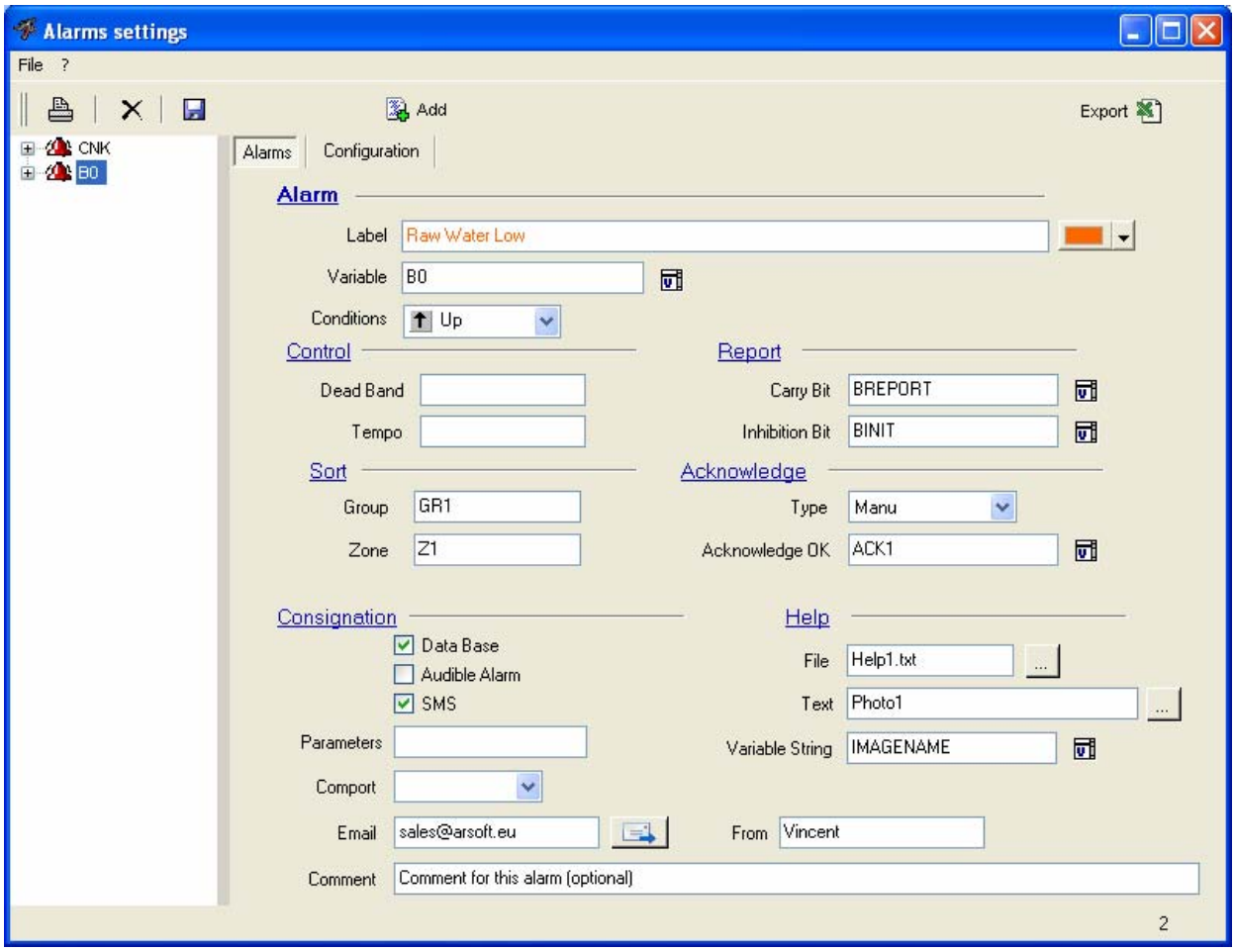


It is in the main menu of Visual I/O editor.

A right click swaps from a list of variable to labels.
The alpha sorting is available in the 2 cases.



Main window



Section Alarm

Label : Alarm text displayed on Screen, Databases, Printer email or serial link.
Variable : Bit or Word to test (alarm bit/word).

Conditions : Alarm Conditions.
Up = On Rising edge of the variable.
Down = On Falling edge of the variable.
>,<,= to a constant or a variable.

Control section

Dead Band : Allows creating a hysteresis zone around the numerical value to test, to avoid successive releases due to the fluctuation. The dead band is valid only on numerical alarms. If the value to be tested is 100 and that the dead band is equal to 4, the global numerical variable can fluctuate from 98 to 102 without retriggering the alarm.

Tempo in ms temporizing the alarm. The alarm is always ON at the end of this tempo. The system will record this alarm. Tempo is to avoid inopportune sets.

Report Section

Bcarry Bit: is a Global bit set (True) when the alarm or one of the alarms is ON.
The same bit can be affected by several alarms.

Inhibition Bit : Bit (optional) to disable the alarm. If this field is empty, the alarm is always at service.

Sorting Section

Group, Zone : Simple text to be treated on a hierarchical basis alarms by membership and allowing a easier sorting.

Acknowledgment Section

The Acknowledge can be automatic or manual.

Automatic : As soon as alarm disappears, it will not be visible any more in all ListBoxes and the date and the hour of acknowledge is recorded.

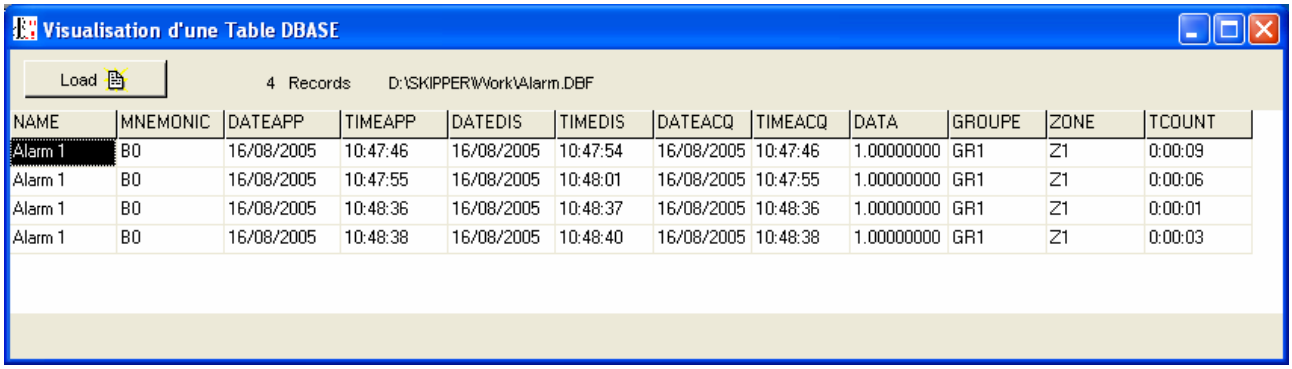
Manual : When alarm disappears the text remains visible in ListBoxes of the form with the bitmap ' *checked Bell'*. The user must click on the text alarm to acknowledge it. The date and the hour of acknowledgment will be recorded.

Consignation Section

Data Base: Alarms are recorded in the file Alarm.dbf (Format DBASE 3+). This file is readable by the standard office software Excel.

Records of a DBase table (Alarm.dbf) :

Is recording: count (TCount) date and hour of appearance, date and hour of disappearance, date and hour of acknowledge and presence time. The other information is the Alarm triggering value (Data) the group and the zone if these are set.



NAME	MNEMONIC	DATEAPP	TIMEAPP	DATEDIS	TIMEDIS	DATEACQ	TIMEACQ	DATA	GROUPE	ZONE	TCOUNT
Alarm 1	B0	16/08/2005	10:47:46	16/08/2005	10:47:54	16/08/2005	10:47:46	1.00000000	GR1	Z1	0:00:09
Alarm 1	B0	16/08/2005	10:47:55	16/08/2005	10:48:01	16/08/2005	10:47:55	1.00000000	GR1	Z1	0:00:06
Alarm 1	B0	16/08/2005	10:48:36	16/08/2005	10:48:37	16/08/2005	10:48:36	1.00000000	GR1	Z1	0:00:01
Alarm 1	B0	16/08/2005	10:48:38	16/08/2005	10:48:40	16/08/2005	10:48:38	1.00000000	GR1	Z1	0:00:03

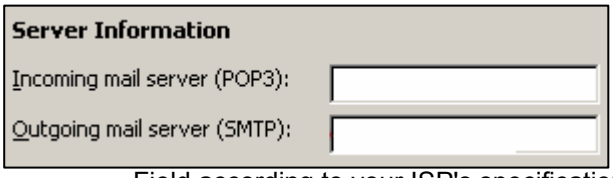
☞ **Note**: *The VisuDBF utility (above is made with Visual I/O) displays and modifies a DBASE table.*

Audible alarm To activate the PC buzzer, when the alarm is present.

Comport To determine the serial comport for sending the alarm text on the serial link.
The serial link configuration (Speed, Data bits etc.) is done in the Configuration tab. .

Email To specify an email address sent to the text alarm.

You must **specify a SMTP server**. This server is specified in Outlook E-Mails accounts.



Field according to your ISP's specifications
In case of Free: the SMTP server will be smtp.free.fr

From Expeditor (Ex here: **Vincent**@webnet.com) if blank the From is **Application**@Webnet.com
Comment Optional text, only for comment.
SMS Allow calling a custom external program. (For example, sending SMS).
This external program will be called only when the checkbox is on.
Parameters In this field you can place a String that will be passed to the external program when the alarm changes state (On, Off, Ack).

Help Section

File It is possible to associate a help file (Text format) with each alarm that will be displayed at runtime, but this file remains optional.
Input in the Editbox the 'File', whose name that you want to associate with the alarm.

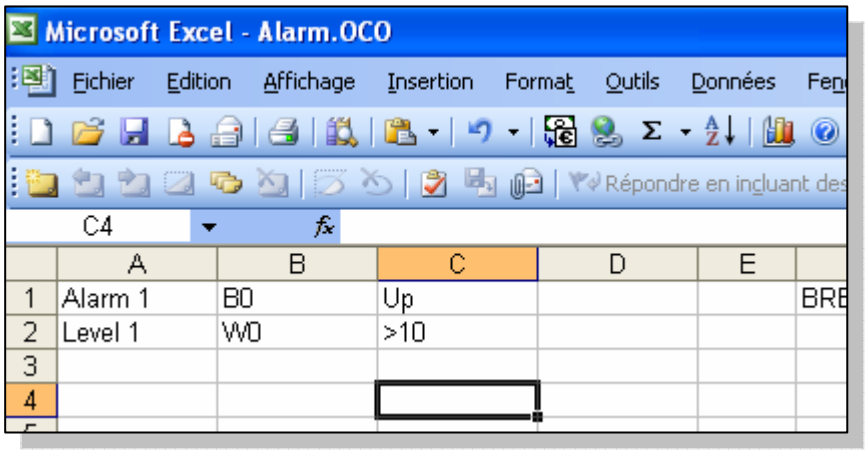
Double-click on the button on the right of this field to call the Windows text editor (NotePad.exe). Strike the Help text then leave NotePad.exe. The specified file is now in the current project directory. If you use Install Builder, manually add these text files (* txt) to the project.

Text It is possible to associate a string receiving a preset text with each alarm.
This text will be downloaded for each selection of an alarm in a ListBox or visualization in a grid of alarms. For example, this text could be a Bitmap name to show a photo for locating the alarm.

In the example above, Photo1 will be downloaded in the String variable IMAGENAME when the user selects the alarm in the visualization grid.

Export to Excel

Click on the Export button. This one opens Excel automatically and displays all alarms in a spreadsheet. The alarms configuration file is ALARM.OCO in the current project directory.



Place a custom program to send SMS.

1/ Each alarm checks the SMS Checkbox.
2/ Specify a parameter that will be received by the program (procedure with parameters) if necessary.

Place the specialized procedure.

The specialized procedure is written in a VPU library or in a form program, you have all the sources for any modification and evolutions at your disposal.
Every alarm will call this procedure if the SMS checkbox is on in the configurator.

To set up the specialized procedure, it is necessary to affect the system variable **CallBackAlarmEx** that is a pointer on a procedure receiving the parameters.

The procedure with its parameters is described below:

Procedure AppelCallBackEx (**Date, Time, AlarmText, Group, Zone**: String; **Treatment** : Integer; **Value**: String; **AlarmType**: Integer; **Parameter**: String);

Date	: Date when the alarm is on. (appearance)
Time	: Time when the alarm is on. (appearance)
AlarmText	: Alarm text.
Group and Zone	: Group and Zone set.
Treatment	: Appearance = 1; disappearance = 2; Acknowledge = 3.
Value	: Value that activated the alarm.
AlarmType	: Bit = 66 ('B') ou Numérical value 78 ('N');

Example to set in a form

```
//**** Procedure calls every change of an alarm *****
Procedure MyCallBackProc (Date, Time, Intitule, Group, Zone: String;
      Treatment    : Integer;
      Value        : String;
      AlarmType    : Integer;
      Parameter    : String);

Begin
End;
{*** Button1 ****************************************}
SObject Procedure Button1;
Var
 PT : Integer;
Begin
   If Start_visu Then
   Begin
     PT:=Addr(MyCallBackProc); //** Callback setup
     CallBackAlarmEx:=PT^;
   End;
End;
```

Send Multiple Emails

You must write a little program to intercept the callback procedure. You can write it in the main form of your application.

```
//* Procedure calls every appearance, disappearance & alarm acknowledge
Procedure MyCallBackProc (Date, Time, AlarmText, Group, Zone: String;
      Treatment    : Integer;
      Value       : String;
      AlarmType    : Integer;
      Parameter    : String);
Var
  Sorte : String;
  Mess  : String;
Begin
   Case Treatment Of
      1 :  Sorte:='Appearance ';
      2 :  Sorte:='Disappearance';
      3 :  Sorte:='Acknownledge';
  End;
  Mess:= Date+' '+Time+' '+Sorte+' '+AlarmText;
  If AlarmType =78 then Mess:=Mess+Value; //*** Numerical alarm

  //** Send 3 emails to differents address
  XSendEmail('My Application ', 'holine@arsoft.eu', 'Alert', Mess, 'smtp.hotmail.com');
  XSendEmail('My Application ', 'Marc@home.com', 'Alert', Mess, 'smtp.hotmail.com');
  XSendEmail('My Application ', 'Marion@Villa.com', 'Alert', Mess, 'smtp.hotmail.com');
End;

{*** Button1 ****************************************}
SObject Procedure Button1;
Var
 PT : Integer;
Begin
   If Start_visu Then
   Begin
     PT:=Addr(MyCallBackProc); //** Setup of the Callback
     CallBackAlarmEx:=PT^;
   End;
End;
```

Send through a GSM

The GSM.Pas and GSM.VPU library allow sending SMS message through a GSM.

Library Interface:

```
Interface
Var
 GSM_COMPORT : Integer=1; //**** Com1 to. Com10

Function Send_SMS (Text, PhoneNumber: String): Integer;
```

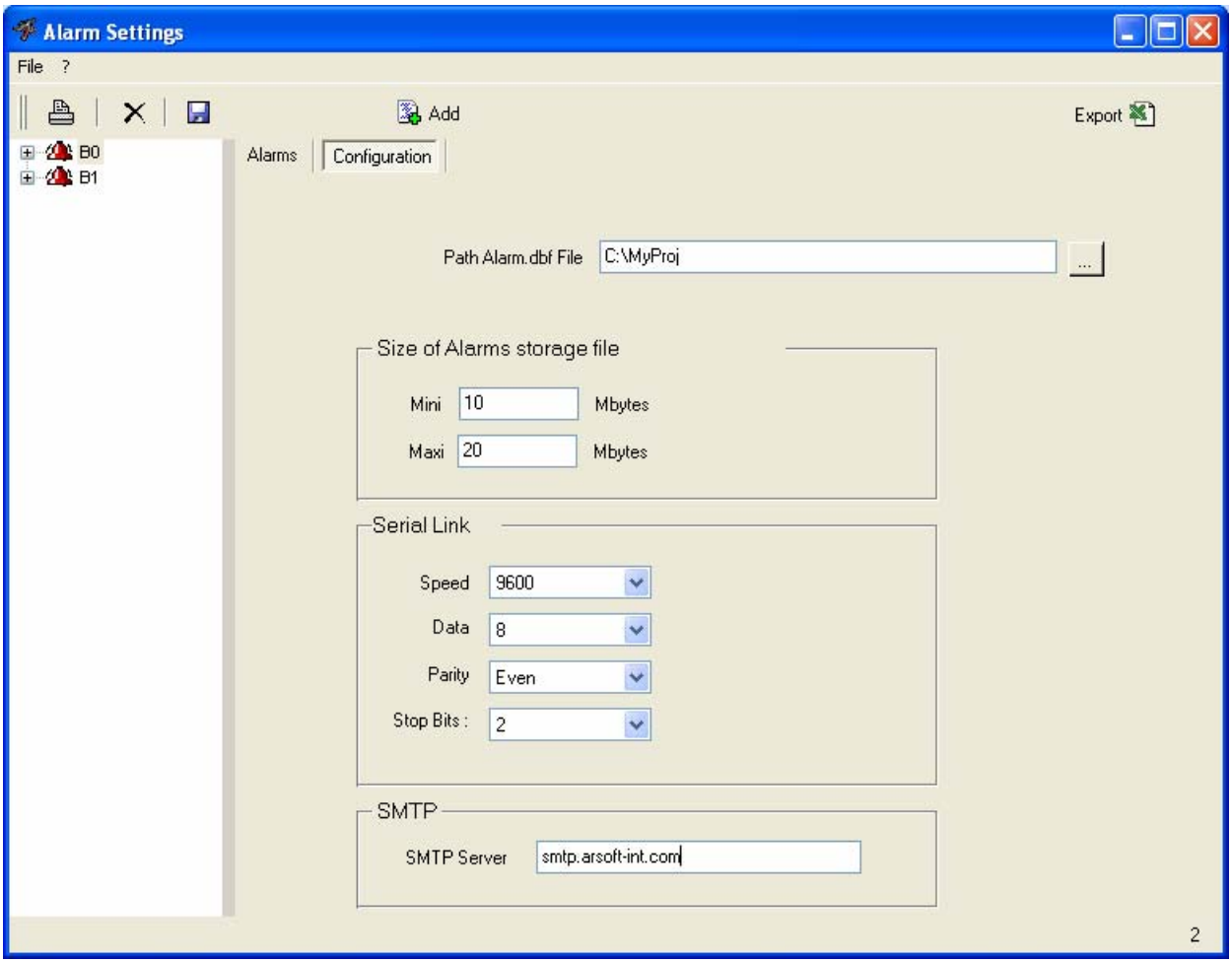
This function sends a text on a telephone number (Phonenumber).
This function returns the number of SMS waiting in the sending buffer of the GSM library.

The library includes a stack for SMS to send, allowing memorizing all the requests in case of buzy transmission.
If a SMS is not sent by the library, the system leaves it in the memory (buffer) and tries to send the next one in the list. The library will come back on the SMS which is not sent later.

```
Uses GSM, Serial;

//* Procedure calls every appearance, disappearance & alarm acknowledge
Procedure MyCallBackProc (Date, Time, AlarmText, Group, Zone: String;
      Treatment    : Integer;
      Value        : String;
      AlarmType    : Integer;
      Parameter    : String);
Var
  Sorte : String;
  Mess  : String;
Begin
   Case Treatment Of
      1 :  Sorte:='Appearance ';
      2 :  Sorte:='Disappearance';
      3 :  Sorte:='Acknownledge';
  End;
  Mess:= Date+' '+Time+' '+Sorte+' '+AlarmText;
  If AlarmType =78 then Mess:=Mess+Value; //*** Numerical alarm
  Send_SMS (Mess, Parameter); //** this procedure is in GSM library
End;

{*** Button1 ****************************************}
SObject Procedure Button1;
Var
 PT : Integer;
Begin
   If Start_visu Then
   Begin
     PT:=Addr(MyCallBackProc); //** Setup of the Callback
     CallBackAlarmEx:=PT^;
     Opencom('1','9600','E','8','2');  //*** Open Comport COM1
     GSM_COMPORT:=1;                   //*** Internal Memory of the comport
   End;
End;
```

☞ **Note** : For another destination of the SMS, it is also possible to redirect regarding the Group or the Zone of alarm.

Configuration Tab



Path for the file Alarm.dbf

You can specify a path for the alarms file (at runtime).
If no file is specified, the current application path is used.

Section of Alarms storage file

To define the minimum and maximum size file of alarms storage (*Alarm.dbf*) and to do an automatic purging.

Maximum in Mega bytes will be the maximum authorised size for the alarm.dbf file. When this file reaches this value, the system will purge the oldest alarms to return to the **Minimum** size specified in Minimum field.

Section Serial Link

Speed, data, Parity and Stop Bits are the serial link parameters used in the section "consignation" in the first tab. All the serial links or the serial link (generally unique) uses these datas.

Section SMTP

SMTP Server: SMTP server is used by the application of your PC. This server allows sending Emails. See your outlook settings.

Technical Datas

Name of the file of alarms parameters setting : Alarm.oco.
Name of the table of the Database of storage of alarms : Alarm.DBF.
Type **Dbase 3+ Table**

Alarm.DBF, Structure of the alarms storage table.

Nom de Champ = *Field name.*
Taille = *Size*
Dec = *Decimals*

C : Chars (80, 20 Chars).

D : Date Type (12/25/01 = 25 December 2001).
N : Numeric (3 = 3 decimals).

Information structure de table dBASE III+ : Alarm.DBF

Description des champs :

	Nom de champ	Type	Taille	Déc
1	NAME	C	80	
2	MNEMONIC	C	20	
3	DATEAPP	D		
4	TIMEAPP	C	8	
5	DATEDIS	D		
6	TIMEDIS	C	8	
7	DATEACQ	D		
8	TIMEACQ	C	8	
9	DATA	N	10	3
10	GROUPE	C	20	
11	ZONE	C	20	
12	TCOUNT	C	8	

Programming on Alarms

The runtime of the applications in peer-to-peer mode is possible (consultation of alarms on several PCs). However, storage should be made generally on one database called the server. To prohibit the data storage on client stations, it is possible to call the procedure **TASK_DB_ALARM** (**Stock** : Boolean);
If **Stock = False**, the functions such as **Storage**, Email, Serial link and klaxon are **disable**.

Disable the general storage of alarms

```
SObject Procedure Circle1;
Begin
  Task_DB_Alarm(TheServer); // If the Server=True, the storage on disk, Email
End;                        // Serial link & klaxon are enable
```

Function TASK_NB_HISTOALARMS : **Integer**;
This function returns the alarms number appeared and disappeared since the beginning or the reset with function TASK_KEEPHISTOALA.

Function TASK_GETHISTOALA (**Index** : Integer) : **String**;
This function returns the alarm from the Index position in the list of alarms.

Function TASK_KEEPHISTOALA (**NB** : Integer) : **Integer**;
This function keeps the X last alarms in the internal history.

Procedure TASK_ACK_ALARM (**Index**: Integer);
Acknowledgment by Index

Example:

```
// *** Get historical ********
For I:=0 to (Task_nb_Histoalarms-1) Do
 Begin
   TS:=Task_gethistoala(I);   //  Get Alarm
   Addstring(TS) ;            //  Add TS into a listbox
 End;
```

Call an owner's procedure at the alarm apparition

Variable CallBackPrintAlarm pointer type (Integer).

This variable acts as a pointer, to specify the final address called by the system on each apparition of a new alarm.
This procedure can process an owner treatment at each alarm.

Internal treatment which the parameter sent to the procedure is formatted like this:

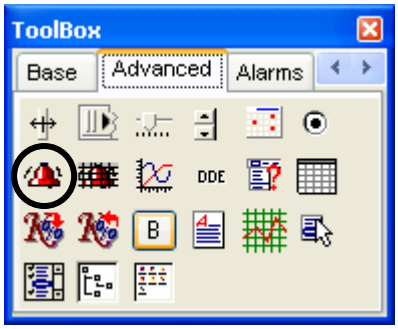
```
Value:=Date+' '+Time+' '+ Label;
AppelCallBack(' '+Value+#13+#10);
```

Example:

```
// *** Call this program by the Alarms manager **
Procedure MyProg ( Value : String);
Begin
  // *** Keep only the label
  While Pos(' ',Value)>0 do
  Begin
    Delete(Value,1, Pos(' ',Value));
  End;
  // *** remove #13 & #10 (CR and LF)
  Value:=Copy(Value,1,Length(Value)-2);
  // *** Value contains the alarm label ****
  If Value='STOP' Then MW10:=MW10+1;
End;

// *** ListBox1 ************************************
SObject Procedure ListBox1;
Var
 PT : Integer;
Begin
  If Start_Visu Then
 Begin
   PT:=Addr(MyProg);
   CallBackPrintAlarm:=PT^;   // *** Affect to MyProg **
 End;
End;
```

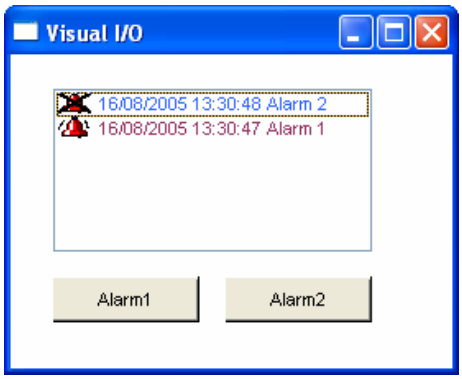
Simple alarms display

Component AlarmList



This component is used to display the present alarms.
It is the simplest way to display the alarms in a form.

At runtime

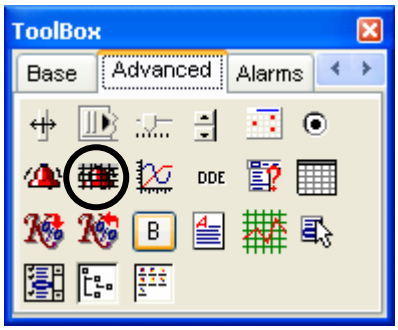


◄ Here 2 alarms are displayed.
The First (Alarm 2) disappeared but is not yet acknowledged by the user.
The second (Alarm 1) is present but not acknowledged.

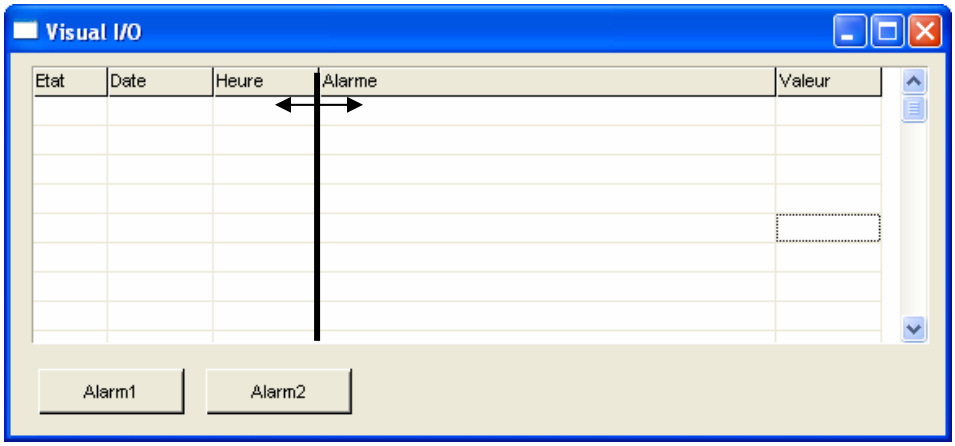
- The bell is to announce the user that the alarm is ON but not acknowledged.
- The checked bell is to announce to the user that the alarm was ON but disappeared.
  The user must acknowledge it to clear the label in the ListBox.
- The label of the alarm without bell announces that the alarm is still ON but is acknowledged by an user or automatically (see alarms setting).

Advanced Alarms display

Alarm grid component



This component enables displaying alarms in a grid.
This grid to display necessary informations is concerned about alarms present.
The component also displays alarms relating to a group and a defined zone and allows modifying or removing the headings of the columns of visualization.



When the component is not selected, you can resize the columns.

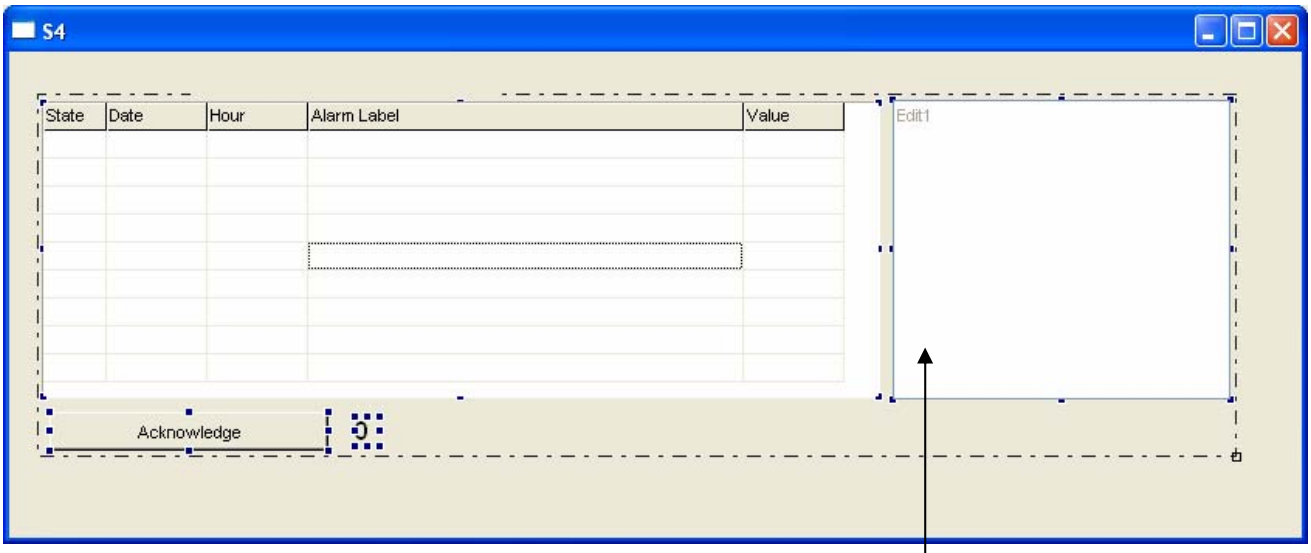
These widths are saved in the form itself.

Component graphical properties

Color contour	$000000
Filling	$FFFFFF
Font	Arial
H.Scroll	
V.Scroll	
Border	False
Ctrl3D	True
Hint	
Group	GR1
Area	ZONE1
State Label	State
Date Label	Date
Hour Label	Hour
Label Name	Alarm Label
Value Label	Value

Group : If no name is specified, alarms of all groups will appear in the grid.
Area : If no name is specified, alarms of all areas will appear in the grid.
State Label : Modifiable, if no name is specified, the column State will be absent.
Date Label : Modifiable, if no name is specified, the Date column will be absent.
Hour label : Modifiable, if no name is specified, the Hour column will be absent.
Label Name : Modifiable, if no name is specified, the label column will be absent.
Value Label : Modifiable, if no name is specified, the Value column will be absent.

Display the Alarm help file



If the component is used alone, the visualization of alarms is done.
If a component **EditBox** is present and **grouped with the grid**, the Help text of the selected Alarm will be displayed automatically at runtime. You must set the properties of the Editbox in **Multilines** mode.

The grid of alarms can display the help text associated in a Multi lines editbox. If a button and a numerical value are present in its periphery, the grid allows also the display of the number of alarms present as well as the acknowledgment of all alarms of the grid.

Place a button and a numerical value, and then group these elements with the grid.

At runtime, the button will acknowledge the block of present alarms and the numerical value will display the number of alarms present in the grid.

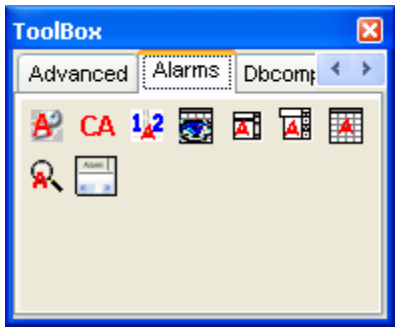
Font	Arial
Multilines	True
H.Scroll	False
V.Scroll	False
Password	False
Border	True
Ctrl3D	True
Hint	
Expert...	

Components for the management of alarms

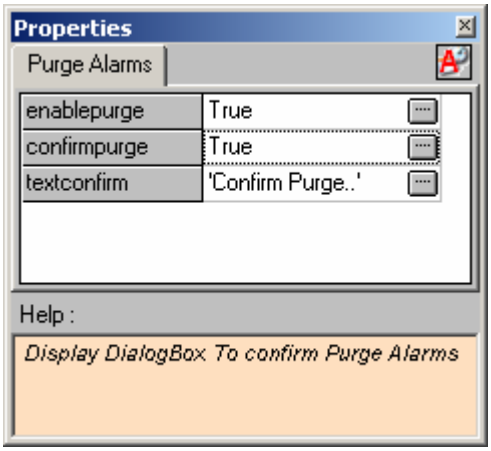
These components are delivered with their sources, to carry out modifications or personalization.

From Left to Right :

PurgeAlarms : Purge Alarms.
Cumul Alarm : Calcul cumul on an alarm.
Choice Date : Display a calendar and to choose a date.
Display Last Alarms : Display the last Alarms.
Listbox Alarms List : Display the labels of alarms.
Combobox Alarms List : Display the labels of alarms.
Display Alarms To Date : Display an alarm at a specific date.
Histo Alarms : History of alarms.
Alarms Extracting : Extractionof alarms



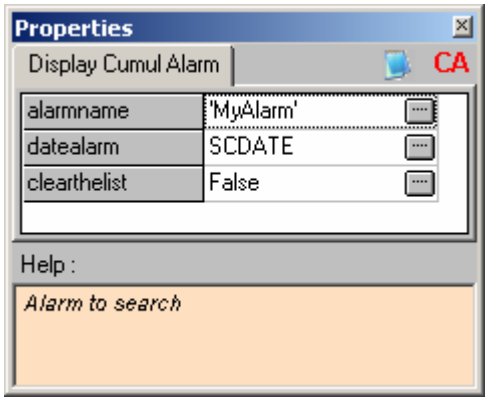
Purge Alarms



Component to display a calendar and to choose a date for purging alarms

EnablePurge : To enable the user to purge the alarms file.
Confirmpurge : When the user select a date of purge, a dialogbox is shown with the text of the cell *TextConfirm*.
TextConfirm : Text to be displayed in the dialogbox in case of purge.
*This component is no more necessary now because it is included in the **Tab Configuration** of the alarms.*
This purge is done when the file (Alarm.dbf) reach a predefined size.

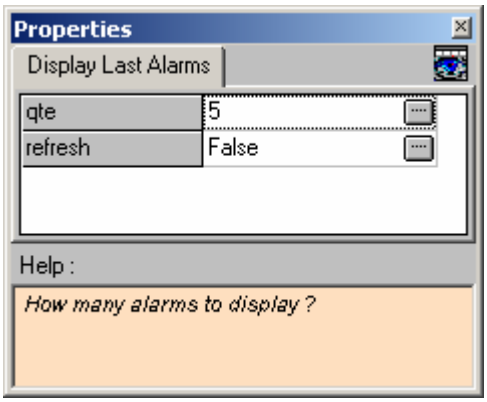
Cumul Alarms



Listbox to display the number of alarms at a specific date.

AlarmName : Alarm name to search.
DateAlarm : Alarm date to search.
ClearTheList : Clear the listbox entirely.

Display last alarms

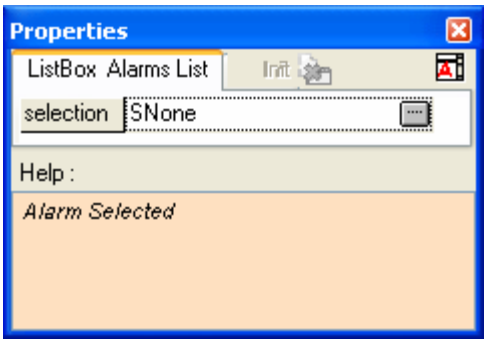


Component to display the X last alarms in a grid.

Qte : Quantity to be displayed.
Refresh : Refresh the grid with the X last alarms.

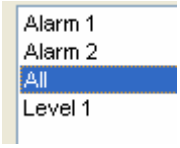
Note: Move the mouse over this component to make an auto refresh at runtime.

ListBox alarms list

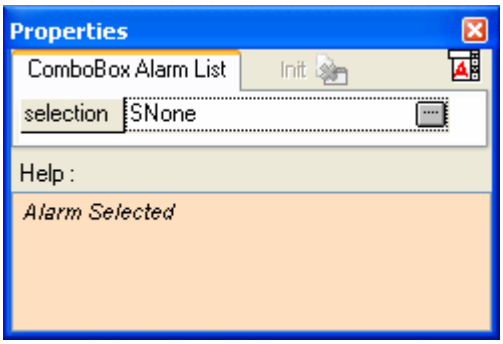


Listbox to display the label of each alarm.

Selection : String variable containing the selected alarm in the listbox.



Combobox alarms list

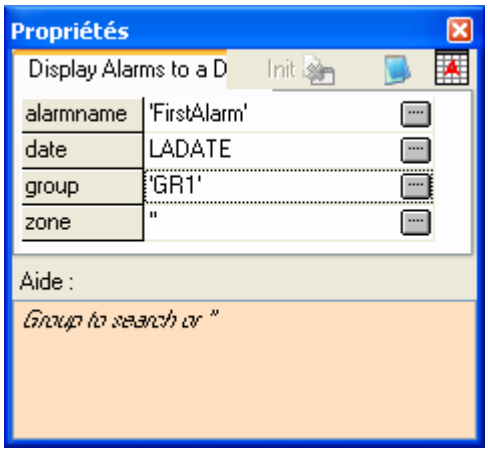


Combo box to display the label of each alarm.

Selection : String variable containing the selected alarm in the combobox.

All the alarm labels are shown.
The text **All** is added by the system, to enable a global selection of the labels.

Display alarm to a date

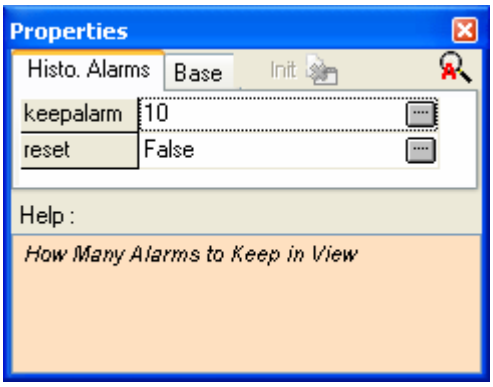


Component to display the apparitions of an alarm at a specific date.

AlarmName : Alarm name to search.

Date : Date of searching
Group : Sorting Group (optional) or empty string.
Zone : Sorting Zone (optional) or empty string.

Histo. alarms



Component to display all the last alarms in a listbox

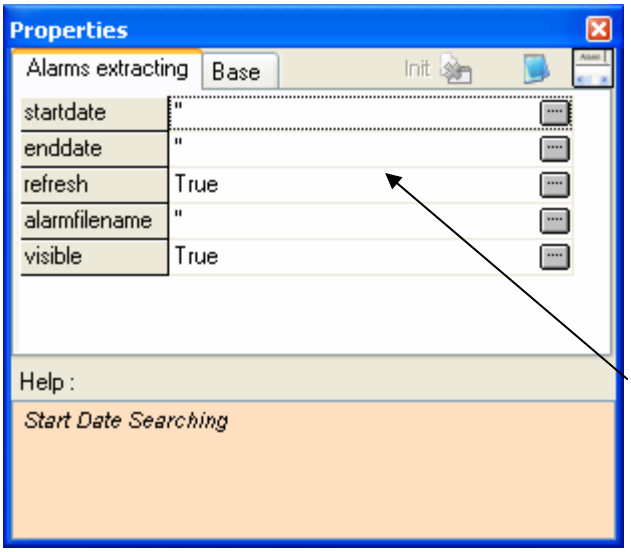
KeepAlarm : Quantity of alarms to display.
Reset : Listbox reset.

If keepAlarm = -1, all the alarms are kept in memory.

Attention to the **Memory overflow**.

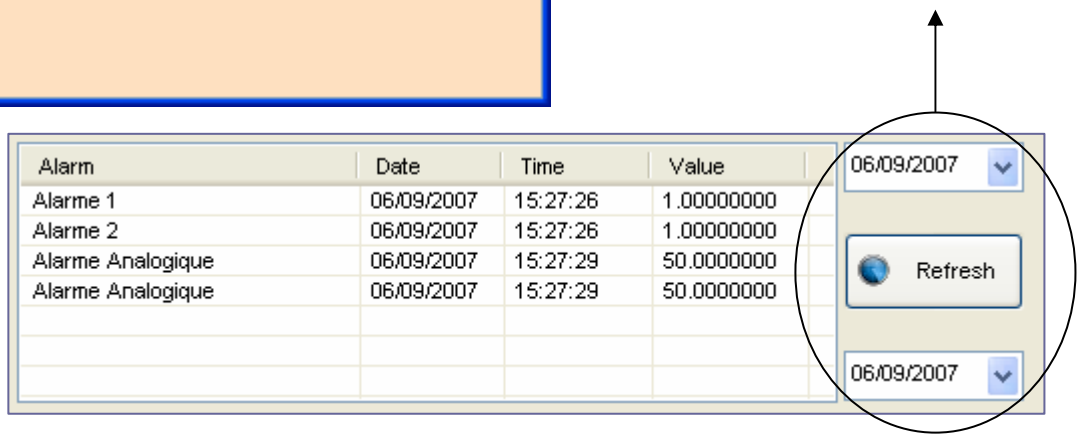
Alarms Extracting

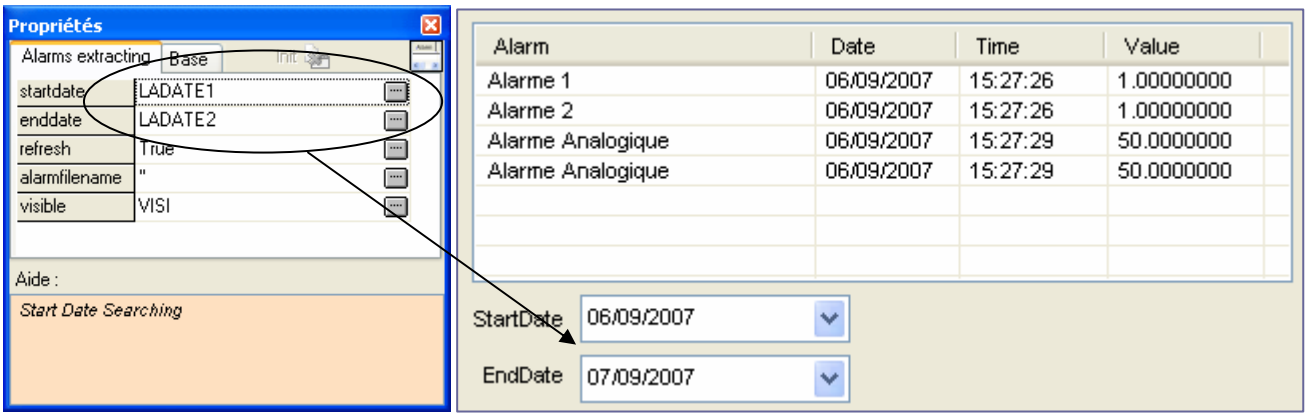
This component allows to show the alarms since a starting date (startdate) until another inclusive date (enddate). If 2 field Startdate and enddate remains empty string (as below) then 2 combobox are display on right of the grid as well as the refresh button. Otherwise these 3 objects disappear for the benefit of 2 string variables containing the 2 dates.



StartDate : Starting date to seek.
EndDate : end date to seek.
Refresh auxiliary command to refresh the grid. Work in parallel with the button.
AlarmFilename : If empty string the component works with Alarm.dbf otherwise use the Path (Ex : "C:\Myproj\Alarm.dbf")
Visible : Enables to Show or to hide the component.

Here Startdate and Enddate are equal to empty strings then 2 combobox are displayed for the selection of these dates.



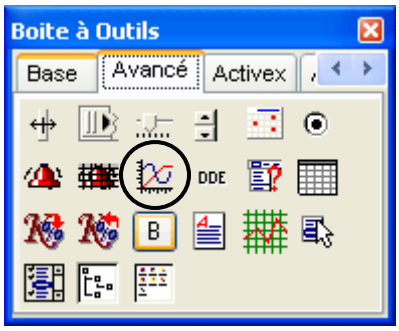


In that case, 2 combobox as well as the refresh button of right disappear. They are replaced by 2 strings variables coming from the application.

Real-time data dynamic plots

Visual IO includes two components allowing plotting data in real time.
These components do not allow recording of the value displayed.

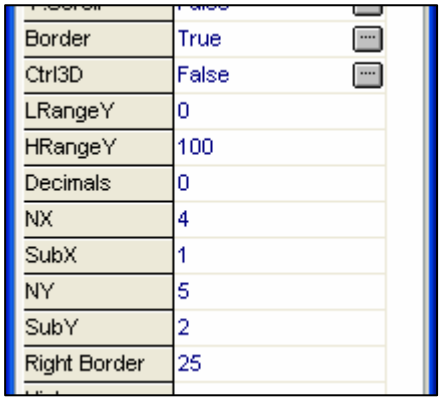
First Component



Name: Trends

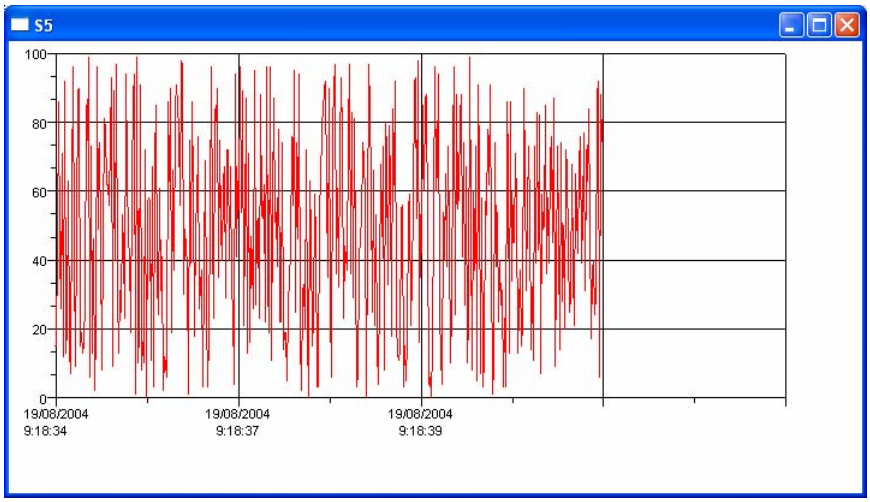
This component enables plotting 8 trend lines in real time. This component does not store any data. The data plotting are performing in real time even this form is not shown on the screen.

Properties box

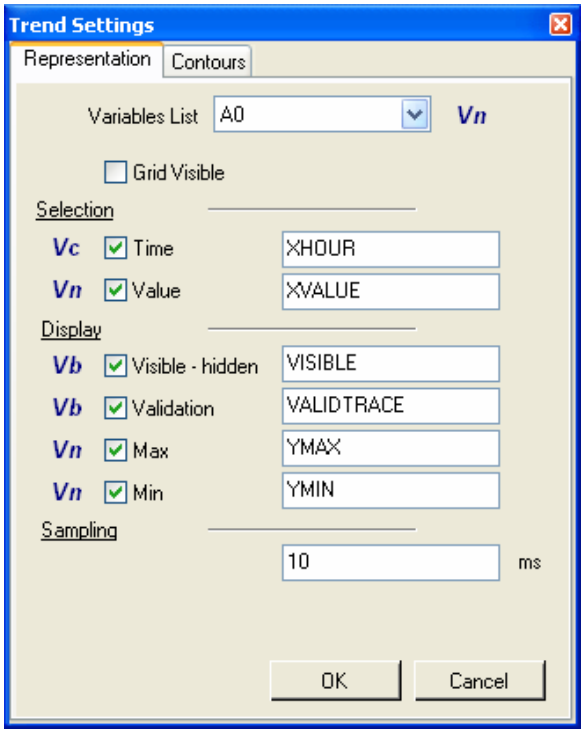


Border	True
Ctrl3D	False
LRangeY	0
HRangeY	100
Decimals	0
NX	4
SubX	1
NY	5
SubY	2
Right Border	25

Border: Specifies the Border Style property for the window. (No visible border or Single-line border).
LRangeY, HRangeY : Minimum and maximum Y-axis sets.
NX, NSubX : Number of divisions and sub-divisions on X-Axis.
Right Border : Space blank at the end of the X Axis (in pixels).



Expert



Grid visible: If Visible is true, the control appears. If Visible is false, the control is not visible.

Time: String variable retrieves the hour pointed by the mouse. When the mouse moves over the graph, this variable receives the hour of the point.

Value : Numerical variable retrieves the value pointed by the mouse.

Visible – Hidden: According to state of Boolean variable, display or hide the trend.

Validation: According to state of Boolean variable, enable or stop the drawing.

Max: Numerical variable allows to specify the maximum value of the Y axis.
Min: Numerical variable allows to specify the minimum value of the Y axis.

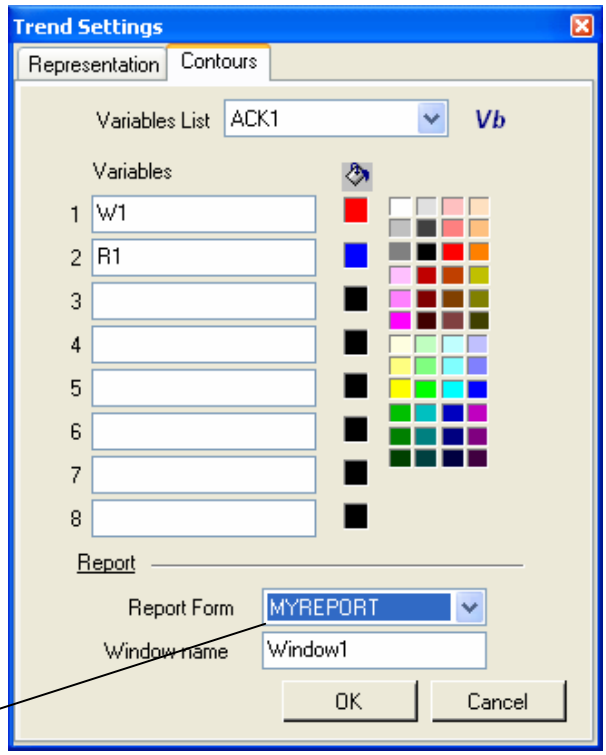
Sampling: Numerical constant specifies the frequency of display points in ms.

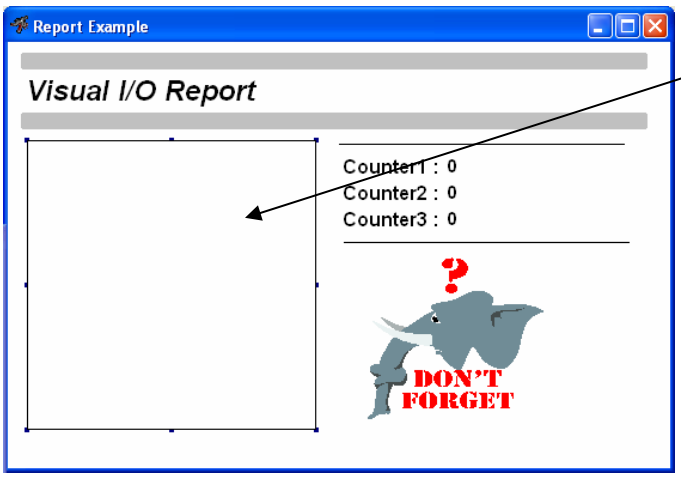
8 Pens are available in the Trends component.

Each pen receives a numerical variable and a color.

Report section:
Visual I/O allows editing sophisticated reports within a framework. It is possible to draw the trend. This framework is drawn into a Window control in a form, which is used as a model for the report.

When the printing occurs with the instruction **PrintReport**, this Window framework is replaced by the layout of the trend.
Specify the name of the Window component of the report in the EditBox *'Report From'*. Also specify the name of the synoptic used as a model of report.



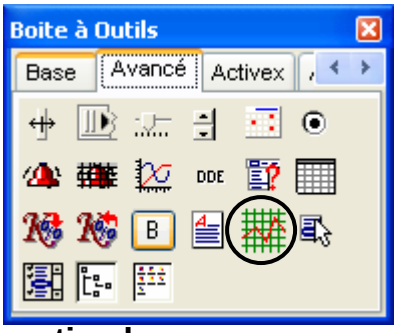


The form is normally drawn, but will not be drawn to the screen. This form acts as a model.

The form will be printed and the window frame will be replaced by the trend present at **PrintReport** instruction time.

The form above is named *MyREPORT* and the window control is named *Window1*.

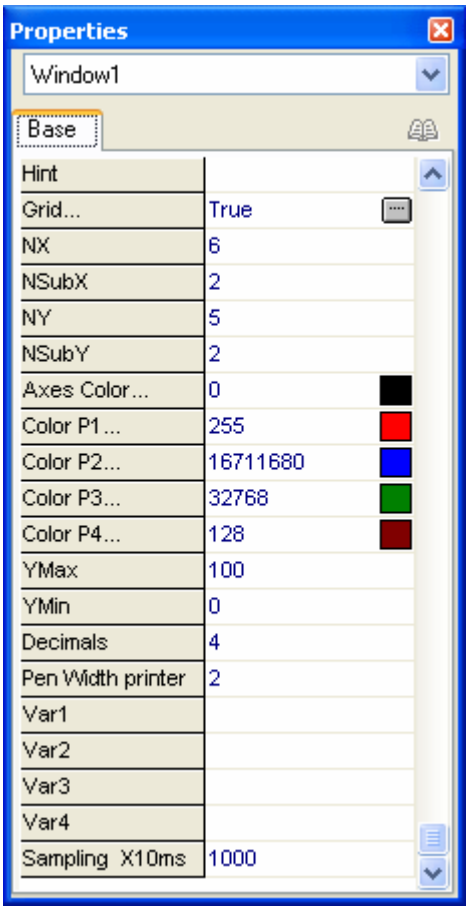
Second Component

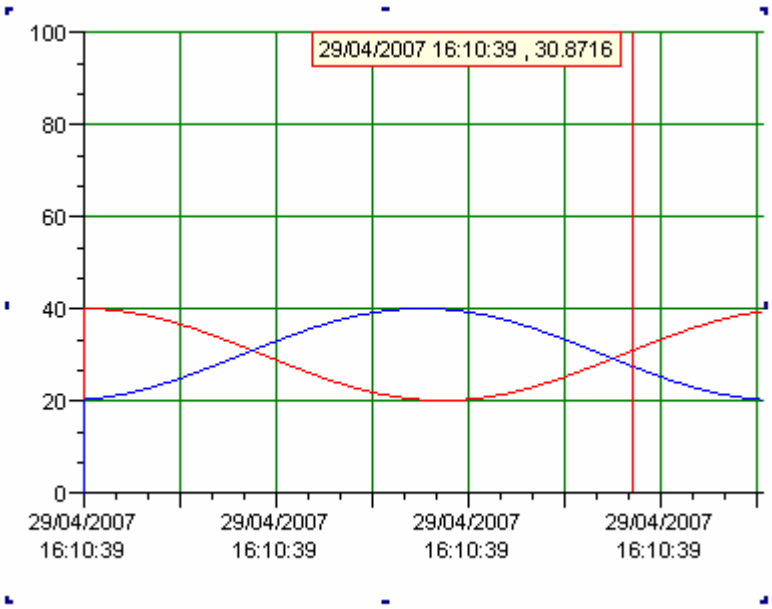


Name: TrendHisto

This component is enabled to display 4 trend graphs in real time. This component does not store any data. The plotting values are done in real time even the form is not on the screen.
This component includes an internal buffer allowing scrolling features to display data before and after the current display.

Properties box





Grid Specifies whether the grid appears in the window.

NX,NSubX,NY,NSubY : Number of minor and major ticks in X & Y axes.

Color P1, P2, P3, P4 Pen color for the trend lines.

YMax and Ymin Minimum and maximum of the Y-axes.

Decimals : Precision of the Y axes.

Pen Width Printer. Pen width for the curves when printing.

Var1 to Var4 : ((optional) Variables names to record and display in real time..

Sampling x10ms : Value to multiply per 10ms for the schedule rate to display and record news values (Var1 to Var4).

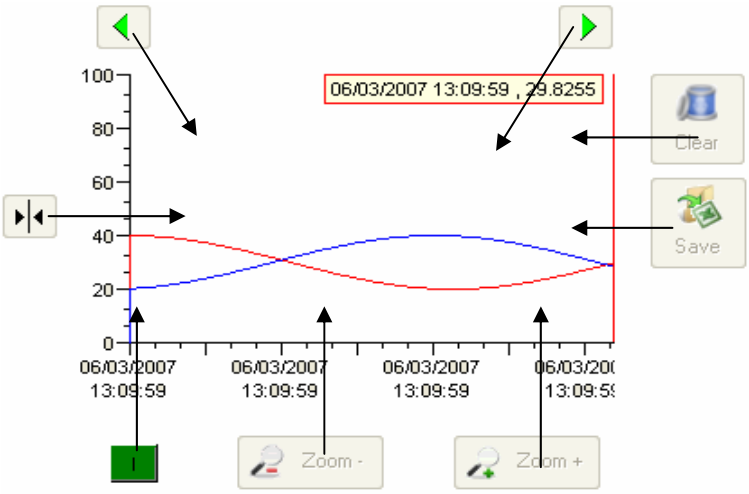
Note: The trend attributes modification is possible by using the Trend component Tab.

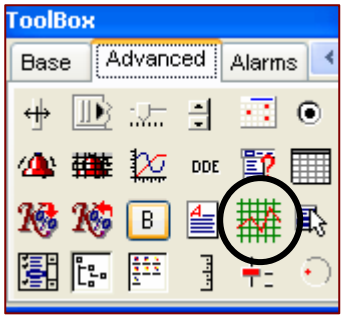
Creating a page including a trend viewer

These components will be used to navigate in an historical file and in a real time data plot.

Trend tab Components

Thanks to peripheral components, this tab allows improving the trend viewer that you will find in the Avanced Tab. If only one trend viewer is used in your form, all components will work with that trend viewer.
If more trend viewers must be used, you will have to group these components with the considerate trend viewer.

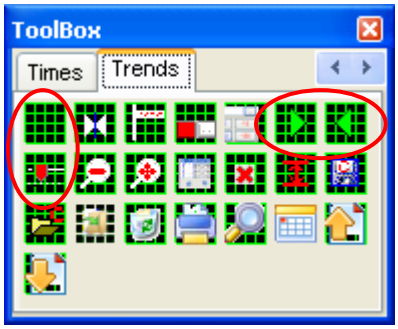


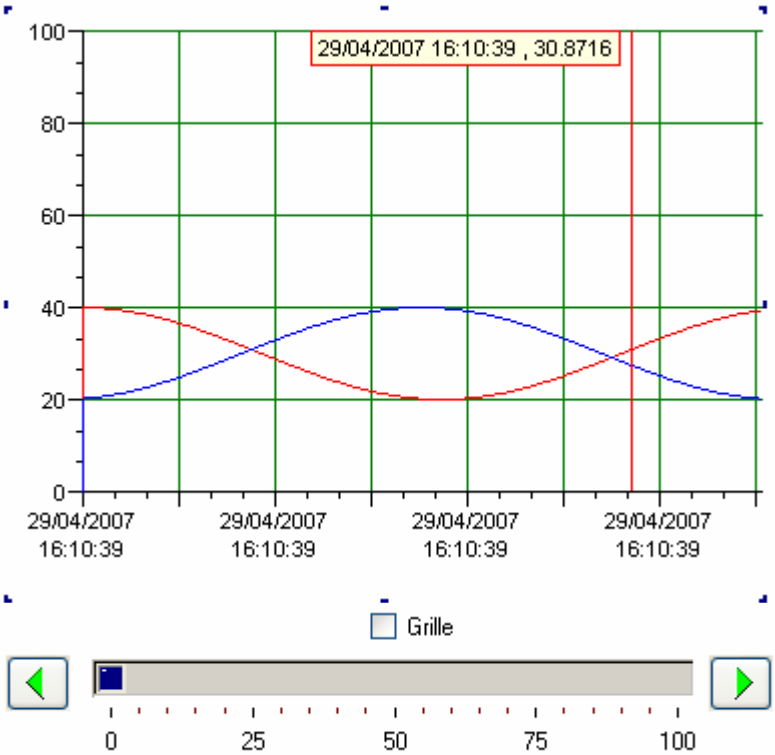


Place this component, then all other peripheral components will work with it.

Navigation in the trend viewer

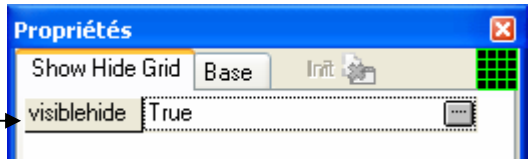
Place a trend viewer (TrendHisto component in the advanced tab).





Components Common parameters :

VisibleHide exits in all components. This parameter is a boolean type (bit) hiding or showing the component in the form.



Display grid

Thanks to a check box, this component displays and hides the grid.

Go to the first point

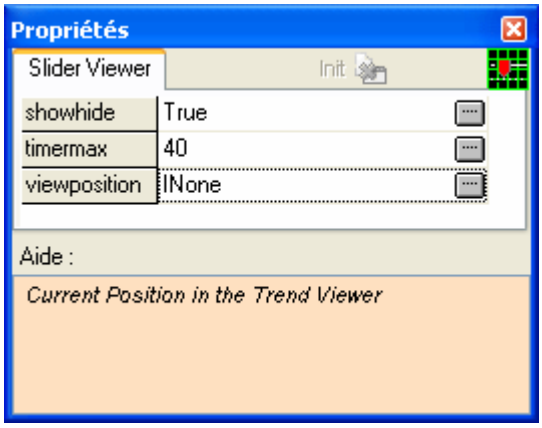
This component is enabled to reposition the trend viewer at the first recorded point.

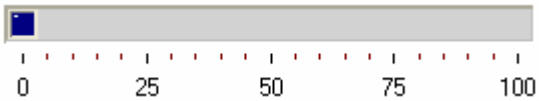
Go to the last point

This component is enabled to reposition the trend viewer on the last recorded point.
This component also is enabled to position the trend viewer in real time trend display.

Navigation through the trend viewer.

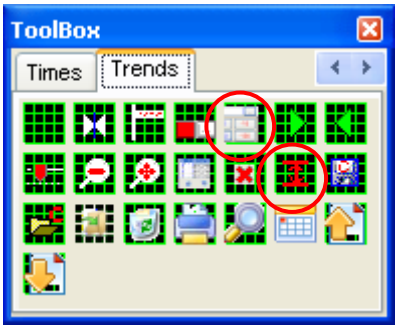
This component is enabled to navigate in the current trend viewer. The trend view incorporates an internal buffer allowing storing an important numerous of points greater than those displayed.
Each trend line has 8400 storable points in the internal FIFO.

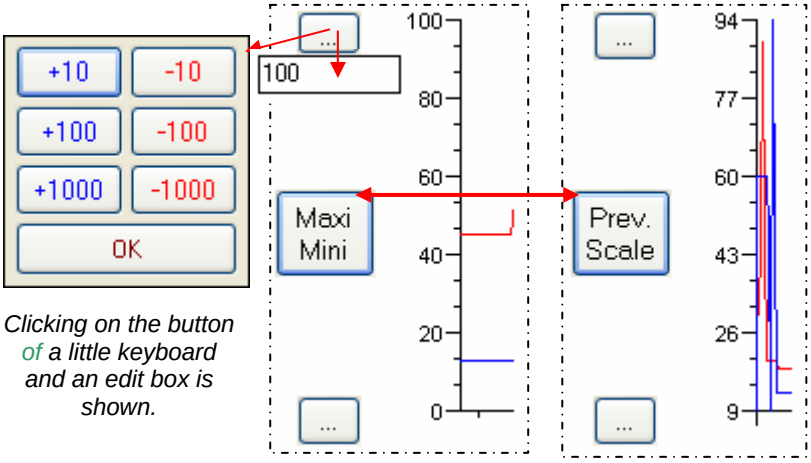




TimerMax: in 1/10sec allows returning the focus to Windows if no more action is made on the potentiometer. After focus returned to (selection) the system, the system refreshes the cursor according to the new records in the trend viewer.
Viewposition: shows the current position in the trend viewer.

Graduations of the Y axis



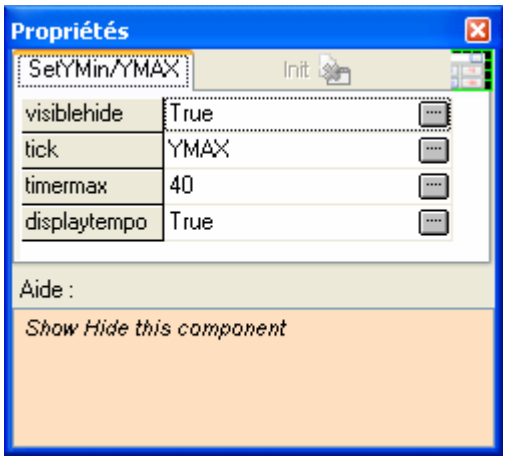


Clicking on the button of a little keyboard and an edit box is shown.

Clicking on the **Maxi/Mini** *button, the scale of auto-adjust to the maxi and to the mini value of the trend lines will be displayed. To click on the button again, the user returns to primary values of the Y-axis.*

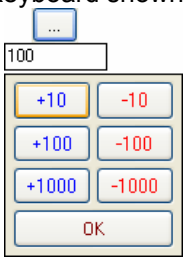
Set the maximum or minimum value of the Y axis

This component allows showing an editbox and a dialogbox with buttons to modify the mini value or maxi of the Y axis.



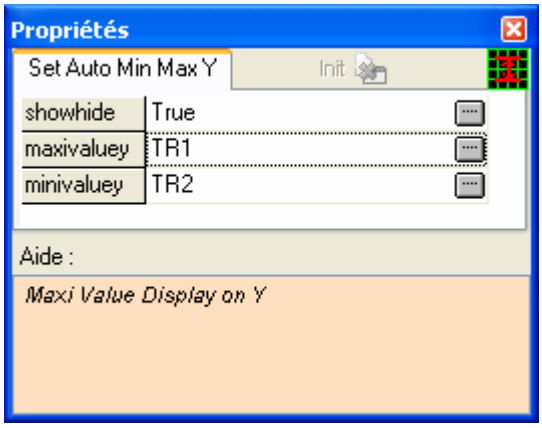
Tick: YMAX or YMIN defines the highest tick or the lowest of the Y axis.
TimerMax: Maximum time before the disappearance of the numeric keyboard.
Displaytemp: If True, the remaining time stays before the disappearance of the numeric keyboard shown on the button. It is only a graphical effect.

The editbox and the little keyboard are hidden at the end of the temporization



Auto adjust the Maxi and Mini values for the Y axis

This component auto calculates the 2 values (maxi and Mini) for the Y-axis.



MaxiValueY & MiniValueY: Report of the 2 values corresponding to the maximum and the minimum written on the Y-Axis.

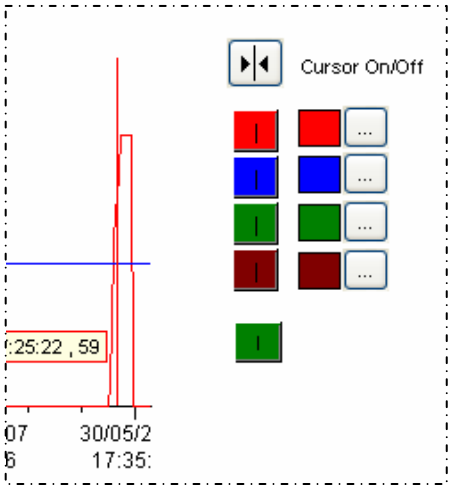
Trend lines cursor and colors

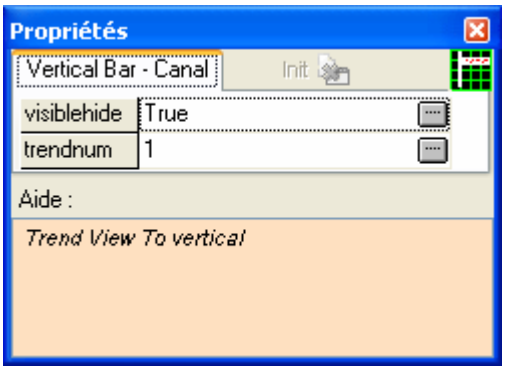
Display the vertical cursor

This component acts as a button displaying or hiding the vertical cursor. This one is moveable with the mouse and indicates the value and date of the point on which the cursor is.

Affect the Vertical cursor to a trend

This component affects a trend line to the vertical cursor.
Moving the cursor will only display the value of the specific trend.
The X-axis will be also graduated with time or numeric value regarding the chosen trend.

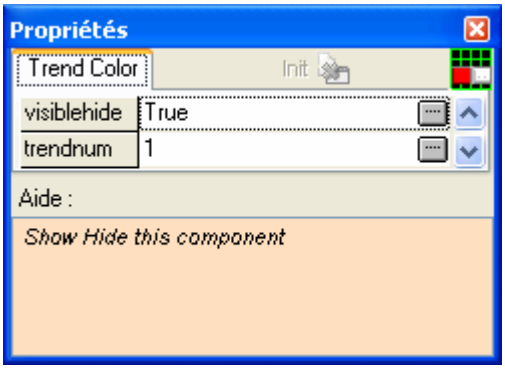




Trendnum: Allows defining the curve number affecting the vertical cursor. If you wish to have a choice of 4 curves, it will be necessary to place 4 different components.

Change the trend line color

This component shows a palette of colors to set a new color affecting a trend line.



Trendnum : Allow defining the trend line number on which the changing color will be made.
Enter a numerical value or a variable in integer type.

Stop / Run internal recording.

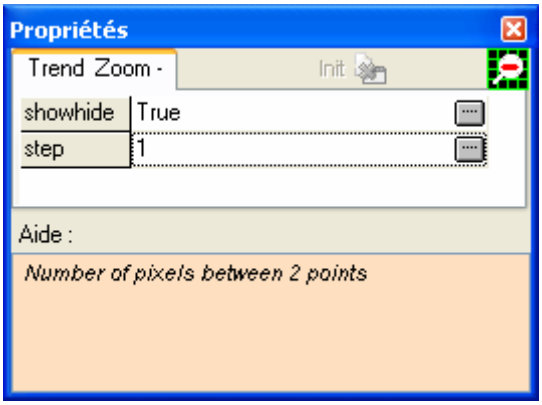
This component allows stopping or running the variables recording (4 maximums) set in the trend viewer.

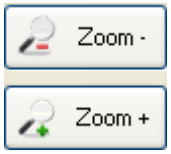
Zooms and multipliers

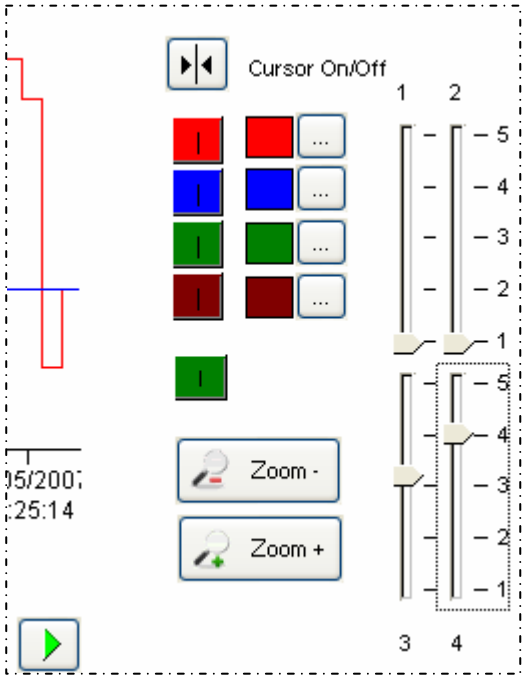
These components perform an expansion of the trends lines for displaying all the details in X.
Another component allows setting a coefficient at each trend line to see correspondences or coincidences in different trends better.

Zoom in and Zoom out.

These components allow enlarging or reducing the space between two displayed points.





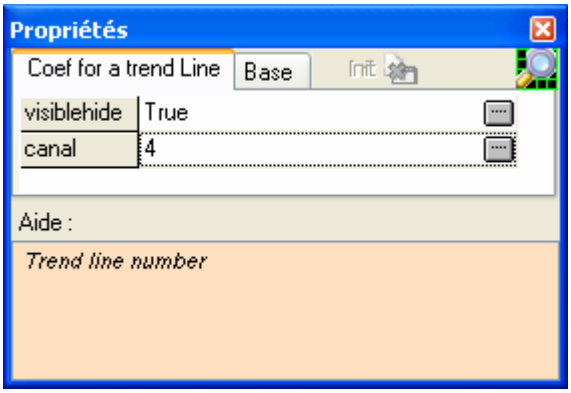


.

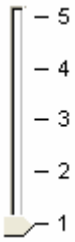
Step: This value is used to increase or decrease the space between points to every click on the button Zoom + or Zoom -.
With Zoom + at every click on the space between points is increased by one step by the value of Step.
The opposite is made for zoom -.

Set a multiplier to the trend values.

This component applies a coefficient to all the values of a specific trend line.



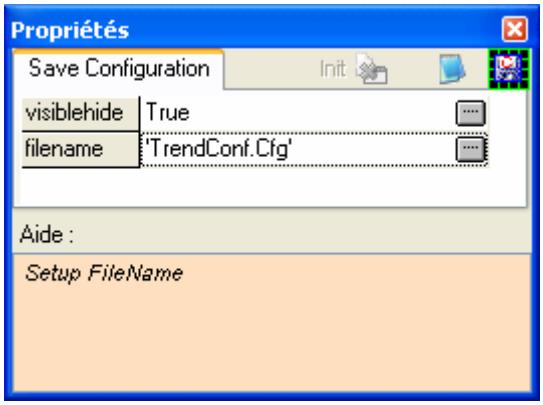
Canal: Trend number affected by the coefficient.
In the Base tab of this component you will be able to modify all the attributes of the slider as horizontal & vertical position, number of ticks colors, etc.



Settings

Save the current setup.

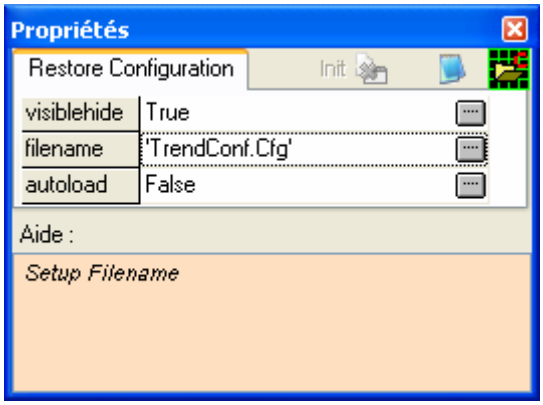
This component saves the current trend viewer set up in run.
Zoom, trend line colors, effect of the vertical cursor of the maxi and mini axis Y are saved.



Filename : File name of the current configuration.

Restoring a setup.

This component allows restoring the current trend viewer setup.



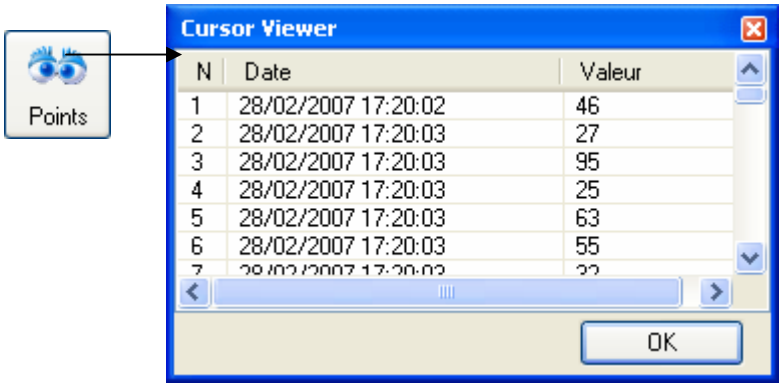
Filename: File Name of the last setup saved.

Autoload: If true, the setup is loaded automatically at the first display of the form.

Display, Save and load points in a trend line

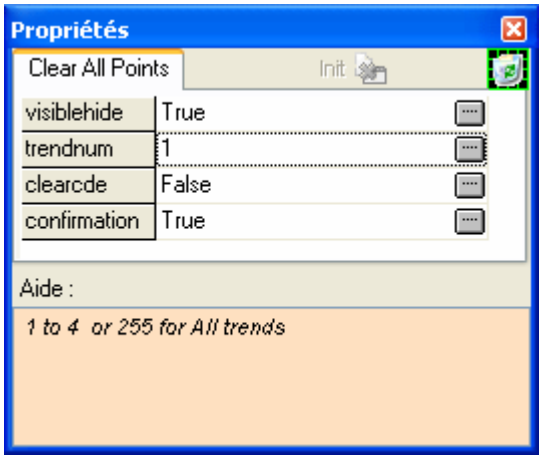
Display points from the cursor.

This component displays a small window shown at the cursor position



Clear all points of a trend line.

This component clears all the points of a specific trend line in the trend viewer. If the parameter "**TrendNum**" is equal to **255,** all trend lines are cleared.



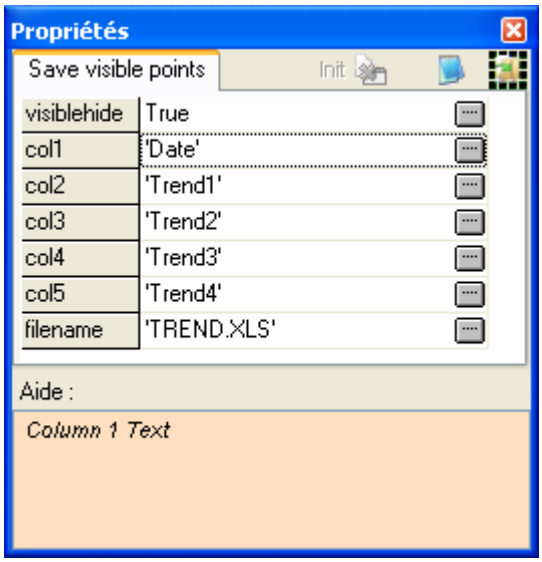
TrendNum: Number of the curve to be cleared (1 - 4). If the value of this parameter is 255, all the curves are cleared.

ClearCde: External and Auxiliary command to clear.

Confirmation: If true, a dialog box is displayed to confirm the deleting points. If false, no dialog box is displayed.

Save visible points in an excel file .

This component saves only the visible points in an Excel compatible file. The file is constituted by 5 Columns. The most left columns contain the dates.



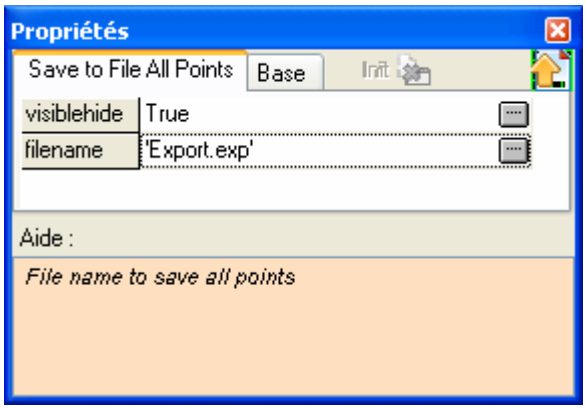
Col1 to Col5: Text of each column for the Excel file generated.

FileName: Excel File name created.

Dates stored in the file are shown on the axis X.

Exportall points into a file.

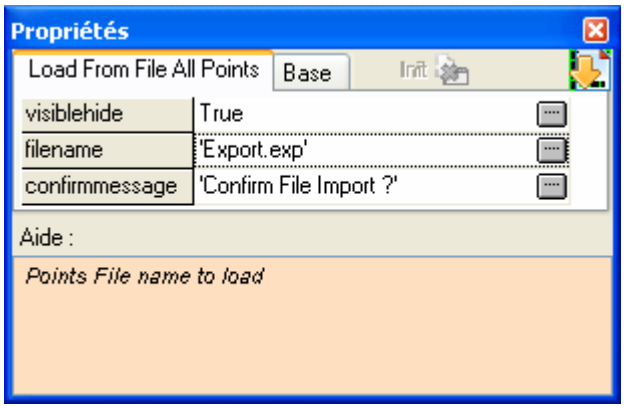
This component saves all the points present in the trend viewer internal buffer to the file specified by Filename. The constituted file is a binary file in a proprietary format. This file can only be read by the other component "import points from a file" describe below.



FileName: File name to save the points.

Import points from a file.

This component imports points from a file specified by Filename.
Before being imported all the points of the trend viewers are cleared. In other words, the viewer is empty.



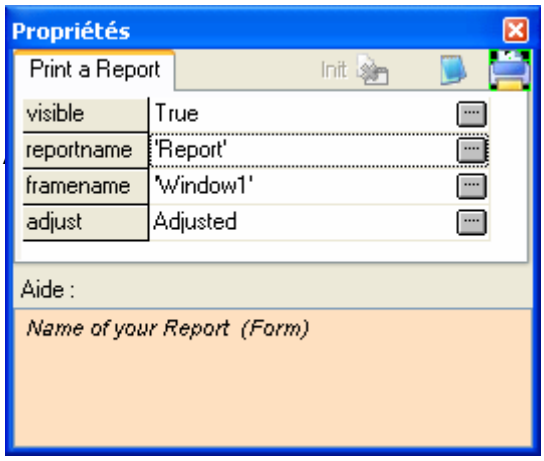
FileName: File name containing the points.

ConfirmMessage: Message is shown in a confirmation dialog box before the component imports the points.
If the message is an empty String, no confirmation dialogbox appears and the import is performed immediately.

Printing trend lines

Print a report including the trend viewer. 

This component prints a Visual I/O report (see chapter) including a trend viewer. The trend viewer frame can exactly draw what is shown in your screen form (adjusted) or use all the points to show a trend viewer much longer than that is shown on the screen form (The printer having more definition than your screen)



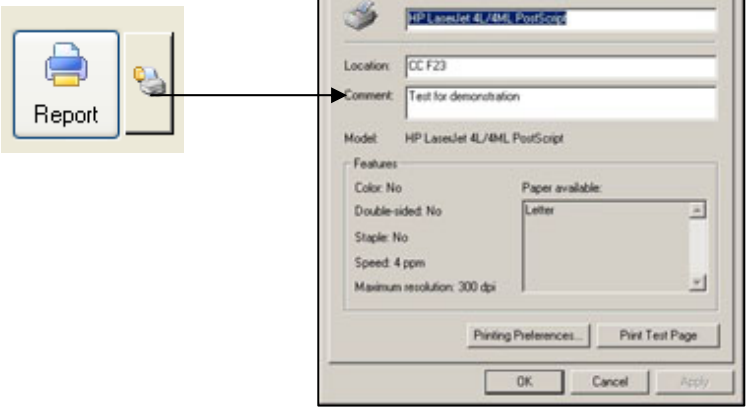
ReportName: Report name including the frame bounding the trend viewer.

Framename: Component name bounding the trend viewer in the report.
Adjust:
Adjusted: Draw the trend viewer of your form in vacant points on the printer.
Large: Draw points quoted by the other one to give a width of trend much more important than that shown on the screen form.

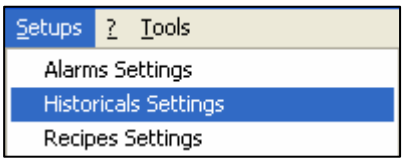
Refer to the chapter report to operate in Visual I/O

Choose and printer settings 

This component configures the printer settings. This component is located in the Files tab.

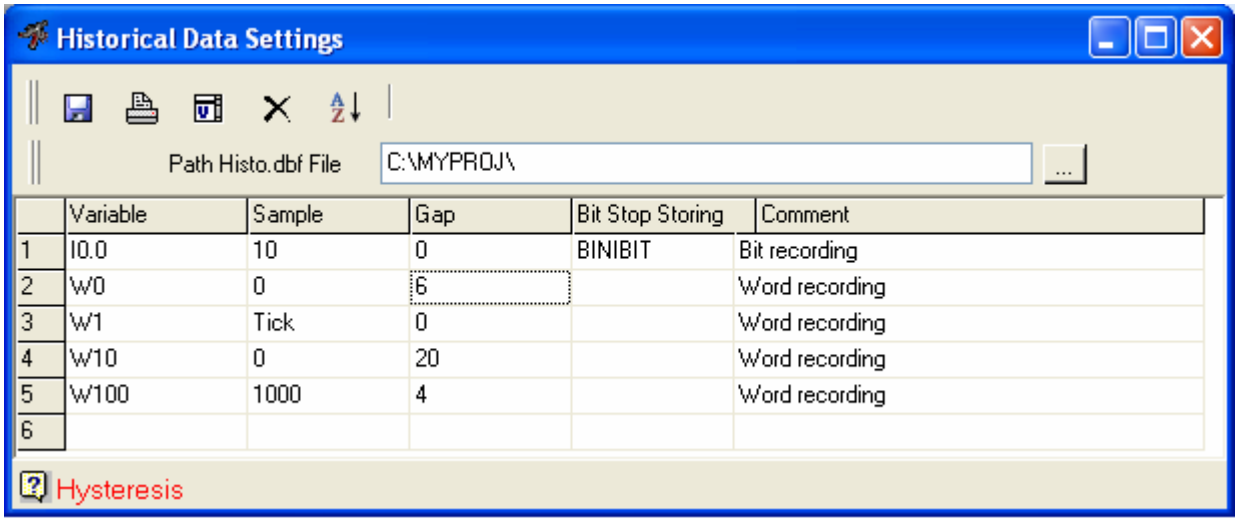


Historical recording and archiving



A configuration grid for the historical recording is displayed to save Boolean or numerical variables (integer, real.). Clicking on the save button, Visual I/O generates and compiles a source file named **PCHISTO.PAS** automatically.
This file is modifiable in the programs section of the editor navigator.

Settings



	Variable	Sample	Gap	Bit Stop Storing	Comment
1	I0.0	10	0	BINIBIT	Bit recording
2	W0	0	6		Word recording
3	W1	Tick	0		Word recording
4	W10	0	20		Word recording
5	W100	1000	4		Word recording
6					

Saving this configuration, a source file **PCHISTO.PAS** is automatically generated.
The compiled file **PCHISTO.VPU** is included in the list *Application Constitution / Priority tasks* located in the menu **Utilities** of the editor.

Variable : Boolean variable or Numerical value to be recorded in the historical file.
Sample : Period in ms (10 ms minimum) or if=0, the recording is done on value changing.
Gap : Significant variation compared to the last value for recording the variable again.
Bit Stop Storing: Optional bit to stop recording the variable.
Comment : Optional comment.

The historical process can be illustrated in different cases as follows:

Case	Sample	Gap
1	0	0
2	0	10
3	10	0
4	10	10

The recording is done :

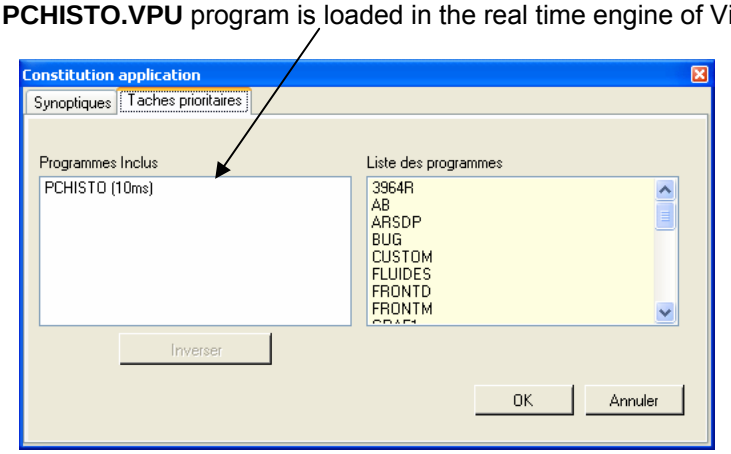
Cas 1 : Only if value is changed.
Cas 2 : When value is changed **AND** if this one is changed at least by 10 points.
Cas 3 : Periodic recording every 100ms only (Time Base).
Cas 4 : Periodic recording every 100ms only **OR** if the value is changed in a variation of 10 points regardless to its previous value. (This case is interesting to limit the size of the historical file).

Other case here:
The I0.0 variable is saved every 10 ms if the Binibit bit is False.

The configuration grid allows to:

Define the historized Variables.
To create a file named Histo.oco (settings).
To compile a program named PCHISTO.VPU that will be called periodically by the application.

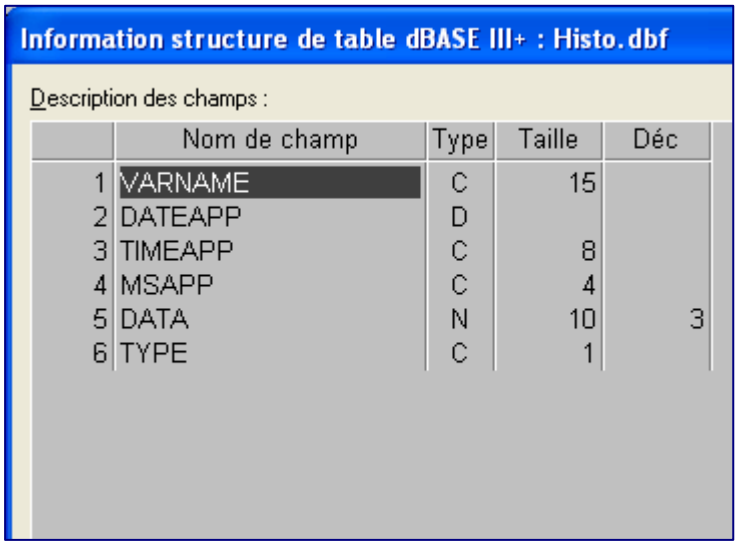
After compiling, the **PCHISTO.VPU** program is loaded in the real time engine of Visual I/O.



Technical Datas

Alarms configuration file name : Histo.oco
Alarms Table Name at runtime : Histo.DBF
Table type Dbase 3+

Histo.DBF, Structure of the history file.



	Nom de champ	Type	Taille	Déc
1	VARNAME	C	15	
2	DATEAPP	D		
3	TIMEAPP	C	8	
4	MSAPP	C	4	
5	DATA	N	10	3
6	TYPE	C	1	

Name : Name of the variable recorded.
DateAPP : Recording Date of the variable.
TimeAPP : Recording hour of the variable.
MSAPP : Recording Milli-Seconds of the variable.
DATA : Value of the variable stored.
TYPE : **B** = Boolean or **V**= Numerical value.

Note The history file is directly compatible with **Excel**.

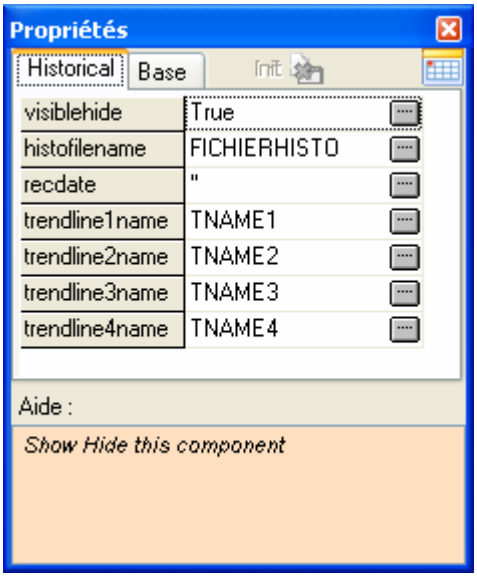
Display historical data in a trend viewer

Introduction

As is seen before, the trend viewer will be used with the Trends tab components. The trend viewer can also display all historical records.

Display an historical file.

Up to 4 data sets (trend lines) may be displayed in one trend viewer. The values are in a historical file specified by the HistoFileName parameter.

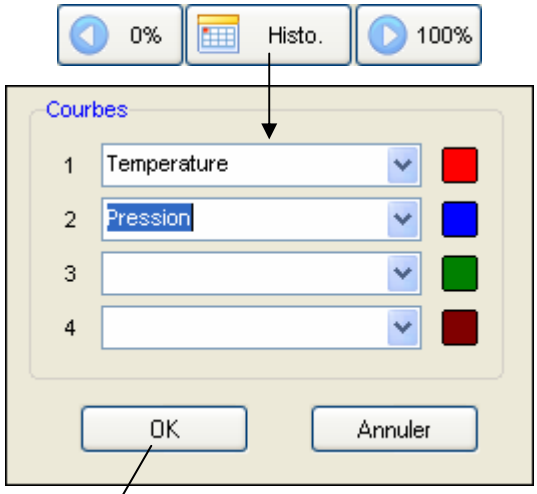


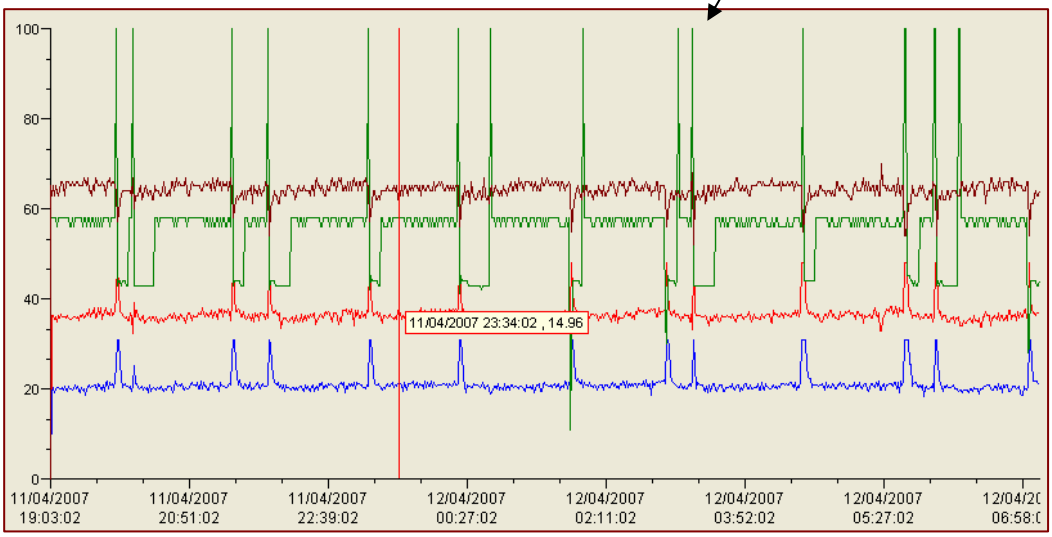
HistoFileName: Historical file name. If this parameter is an empty string, the search is performed on the current historical file.

Recdate: Starting date to search.

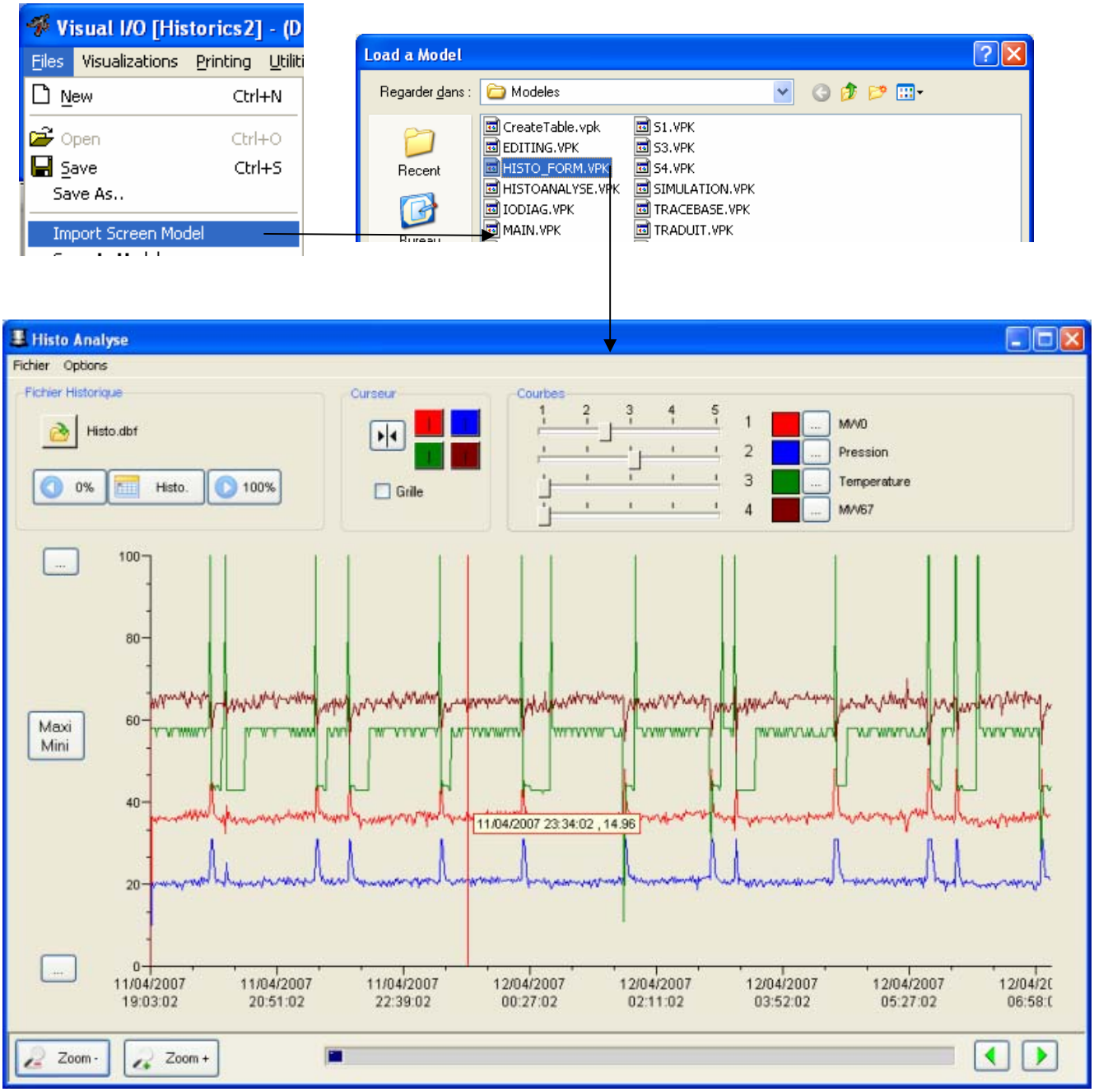
Trendline1Name to Trendline4Name: Report to string variables retrieving the name of the variables chosen by the user.

By clicking the histo button. The user can choose 4 names of variables to draw.
By clicking left and right buttons, the operator can navigate in the historical file if this one is important.
The left button indicates the position of the first visible point in the file and the right the position of the last point.
If the values are 0 % (Left button) and 100 % (right button), the file is completely visible in the trend viewer.





Ready-to-use screen model for history management

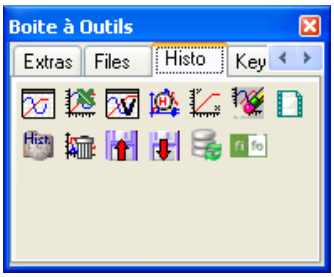


The HISTO_FORM model allows the consultation of history. You can incorporate this form directly into your project without any new declaration of global variables.

Note: For printing see chapter trends printing.

Components for historical management

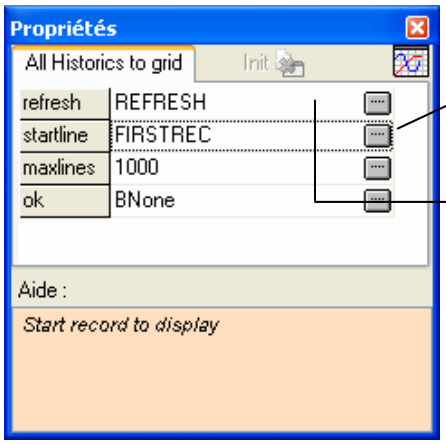
These components are delivered with theirs source code, allowing modifications or customizations.

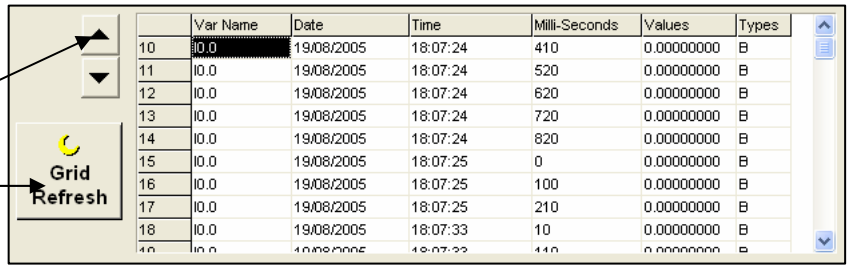


From Left to Right

AllHistoToGrid: Display of the historical file in a grid.
ExportHistorics: Export of historical to Excel.
VarHistoToGrid: Display a specific history to a grid.
MCHisto : Historical search and display.
TrendXY : Display of an array of points.
PurgeHistorics : Purge of the historical file at a specific date.
ShowEvents : Display events (see chapter events).
MCHistoEx : Advanced historical search and display.
ClearHistorics Clear the historic file (Histo.dbf).
StoreVars Save a series of consecutive variables in a file.
LoadVars Load a series of consecutive variables in a file.
HistoFiFo Limitation of the Histo.dbf file size by creating a fifo on the disk
FifoValues Create a FiFo for Global Variables.

AllHistoToGrid





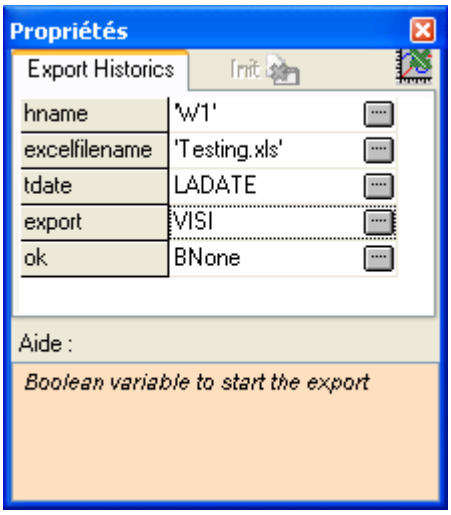
The **ALLHistotoGrid** component displays the whole recordings of the Histo.dbf file in a grid. Here the UP/Down top left component to scroll the StartLine parameter which enables moving the visualization in the grid.
This component does not carry out selection on the displayed histories.

Refresh: Variable to refresh the grid by the containing of the file Histo.dbf.
Ok : Result of the refresh command (reset by program).
StartLine: First record to be displayed in the grid.
MaxLines: Number of records to be displayed in the grid from startline.

☞ **Note**: To modify *Startline*, the user can navigate in the historical file (Histo.dbf).

ExportHistorics 

The ExportHistorics component exports in a text files formatting in Excel format. The historical of a specific variable is up to a specific date.

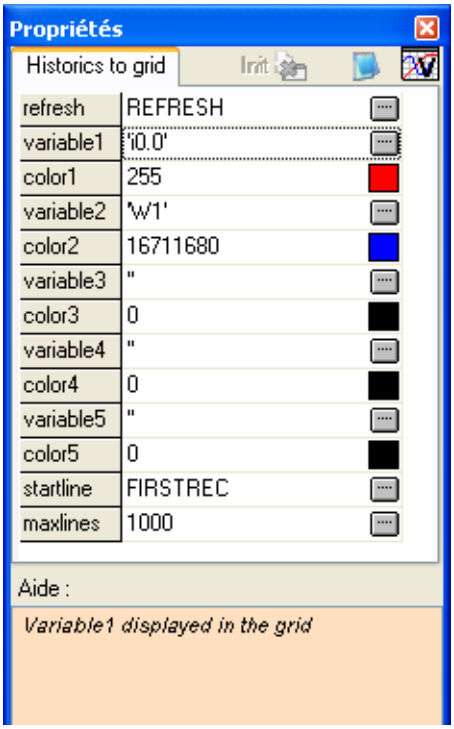


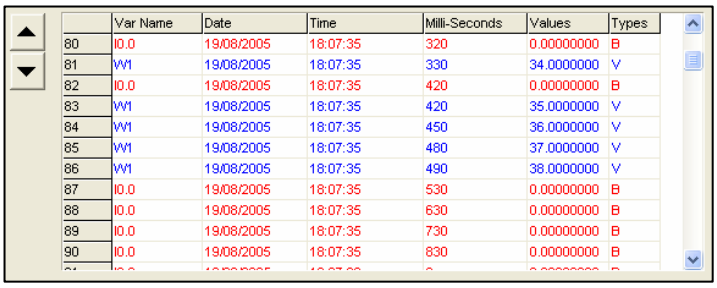
HName: Historized variable name to be exported.
ExcelFileName: Excel file name to be created.
TDate: Maxi Date to export the variable. All the anterior or equal records to this date will exported in the excel file.
If TDate=", the system takes the current date automatically.

Export: Command Variable to be exported.
Ok: Result of the export. The component returns True if almost one export is done.

☞ **Note** : For the choice of date, you can use the ChoiceDate component in the Alams tab.

VarHistoToGrid 





	Var Name	Date	Time	Milli-Seconds	Values	Types
80	I0.0	19/08/2005	18:07:35	320	0.00000000	B
81	W1	19/08/2005	18:07:35	330	34.0000000	V
82	I0.0	19/08/2005	18:07:35	420	0.00000000	B
83	W1	19/08/2005	18:07:35	420	35.0000000	V
84	W1	19/08/2005	18:07:35	450	36.0000000	V
85	W1	19/08/2005	18:07:35	480	37.0000000	V
86	W1	19/08/2005	18:07:35	490	38.0000000	V
87	I0.0	19/08/2005	18:07:35	530	0.00000000	B
88	I0.0	19/08/2005	18:07:35	630	0.00000000	B
89	I0.0	19/08/2005	18:07:35	730	0.00000000	B
90	I0.0	19/08/2005	18:07:35	830	0.00000000	B

Refresh: Command variable to refresh the grid with the containing of the Histo.dbf file.

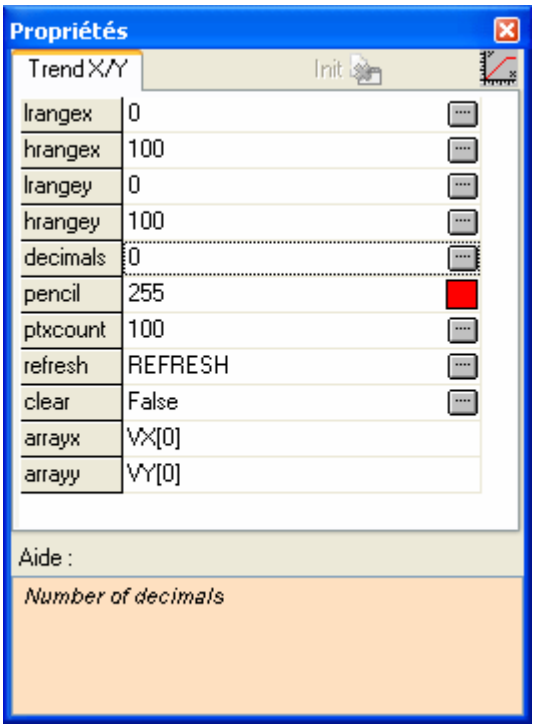
VariableX : Name of the 5 historized variables to be displayed in the grid.
ColorX : Text color of the found variable.
StartLine: First record for which the system search in the historical file (File Histo.dbf).

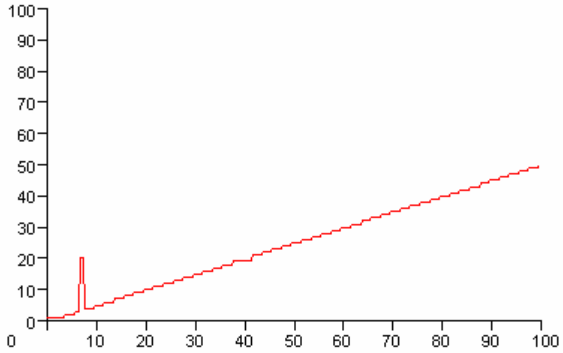
MaxLines : Maximum records to be scanned in the historical file (Histo.dbf) and this to the beginning specified in startline.

☞ **Note** : To modify *Startline*, the user can navigate in the historical file (Histo.dbf).

TrendXY

The TrendXY component displays a table of points in Y comparing to a table of points in X. It is necessary to initialize two arrays of POINTS X and Y and to specify the number of points to be displayed in the component.

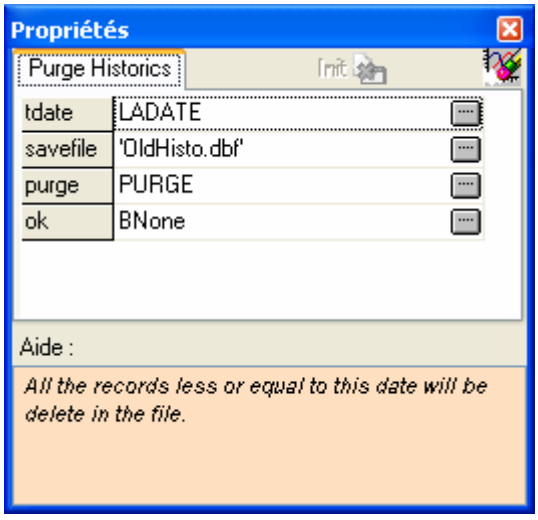




LRangeX : Minimal value of the X axis.
HRangeX : Maximal value of the X axis.
LRangeY : Minimal value of the Y axis.
HRangeY : Maximal value of the Y axis.
Decimals : A number of decimals displayed for the X and Y axis
Pencil : pencil color.
PtxCount : A number of points (values) in the X Array.
Refresh : To refresh the trend.
Clear : To clear the trend.
ArrayX : Array of Values in X. (Type Array[0..X] of Real)
ArrayY : Array of Values in Y (Type Array[0..X] of Real)

Purge historics

Component PurgeHistorics purges the Histo.dbf file until a specified date.
The deleted records are saved in a backup file with .dbf extension (Dbase).



TDate : Date limit to erase the records.
All the records lower or equal to this date will be cleared from the file.

SaveFile : File name for the deleted records.

Purge : Start the Purge.

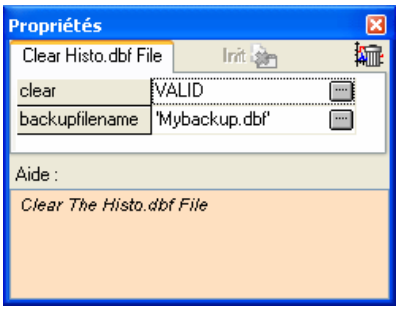
Ok : Purge result.

Return true if deleted records is performed.

The PurgeHisto function which is used by this component is available in the file ObjHisto.Vpu.
To create the Excel file in a different directory from the application, it is necessary to specify the complete path of the file.
Attention: The component is searching for a specific date (Tdate) to begin the purge.
If the date is not found in the histo.dbf, No Purge is done.
If you want to clear and archive the histo.dbf, use the Clearhistorics component because this component does not search for a date.

ClearHistorics

The ClearHistorics component enable copying the current Histo.dbf file into the parameter BackupFilename if this one is not an empty string. After backup, the current file Histo.dbf is cleared.

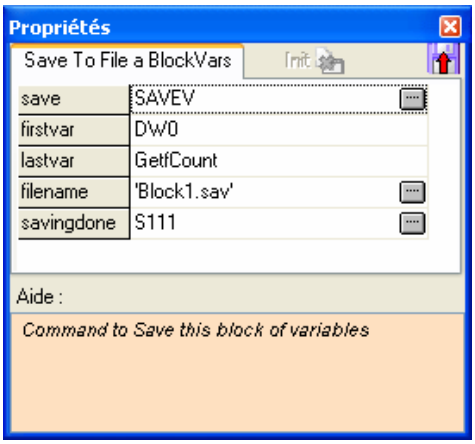


Clear: Save the current Histo.dbf file if the *Backupfilename* parameter specify a file name, then erase all the records in the current histo.dbf file.

BackupFilename: Back up file name created before clearing the histo.dbf records.

StoreVars

This component enables saving a global variables memory mapping in a file quickly, beginning at FirstVar and finishing the variable declared before the variable is specified by LastVar.



Example of Global variables.

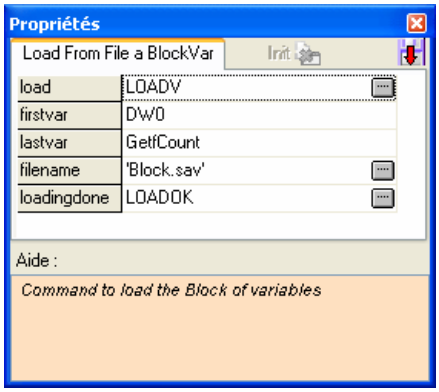
```
DW0 :  Integer; //*** First variable
DW1 :  Integer;
DW2 :  Integer; //*** Last variable
GetfCount :  Boolean; //** Lastvar+1 not saved
```

Save:The save command.
FirstVar: Specify the first variable to save.
Lastvar: Specify the last variable +1 to save. This variable is only for calculation, but is not saved.

FileName: Filename for saving the variables.
SavingDone: Indicate whether the file was created or not.

LoadVars

This component enables loading a global variables memory mapping from a file quickly, beginning at FirstVar and finishing the variable declared before the variable is specified by LastVar (complement of Storevars)



Example in Global variables.

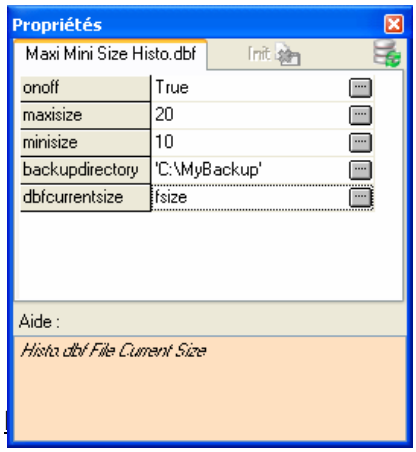
```
DW0 :  Integer; //*** First variable
DW1 :  Integer;
DW2 :  Integer; //*** Last variable
GetfCount :  Boolean; //** Lastvar+1 not saved
```

Load: The loading command.
FirstVar: Specify the first variable to load.
Lastvar: Specify the last variable +1 to load. This variable is only for calculation, but is not loaded.

FileName Source file name.
SavingDone: Indicate whether the file was loaded.

HistoFiFo

This component enable setting the size of the historics file (Histo.dbf). The size will grow from a minimum to a maximum. These 2 values are multiplied by 100KBytes.



OnOff: Enable or disable the Fifo control on the current histo.dbf file.
MaxiSize: Maximum size of the Histo.dbf file x 100KBytes
MiniSize: Minimum size of the Histo.dbf file x 100KBytes

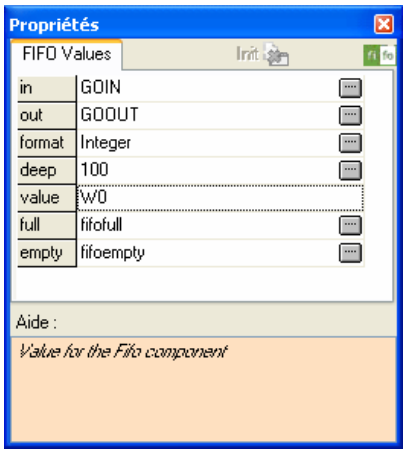
BackupDirectory: Indicate whether the string is empty or not. The purged data is copied to Directory. The name of the purging file is the date with .DBF extension (format DBase). The dbase format can be read by Visual I/O as well as Excel.

DbfCurrentSize: Optional returns the current histo.dbf file size.

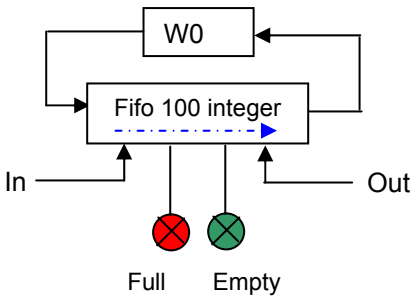
Backupdirectory can be a constant as above which is also a variable containing the path of the directory when the purged values of the historic file which will be stored under a name formated as follows:
Date . DBF ex : for the date 12/03/2006 the filename will be 12_03_2006.DBF. If it is necessary to purge several times a day, the system will create 12_03_2006[2].DBF, 12_03_2006[3].DBF and so on.
If Backupdirectory is an empty string, no backup file will be done.

FiFoValues

This component allows creating a FIFO stack (First in First Out). The First entered is the first to be taken out. This stack allows storing values and restoring them.



In: Command for entering a new value into the FiFo.
Out: Command for exiting a value from the FiFo. The value is the older value entered.
Format: Format of the variable.
Deep: Number of value in the FiFo.
Full: True if the Fifo is full.
Empty: True if the Fifo is empty



Programming on historical files

To known the current size of the historical files

In the OBJHISTO VPU, variable FILEHISTOSIZE indicates the size in bytes (Byte) of the histo.dbf file.
To use this variable in a program, do not forget to declare **Uses ObjHisto**;

Purge the historical file by program

```
Uses sysplc,Windows,OBjHisto,Microbase;
// *** Value2 ************************************
SObject Procedure Value2;
Var
  R : real;
Begin
 SetValue(FileHistoSize);           // *** Display the size
 If FileHistoSize>10000000 Then     // *** Size of the histo file >10 Mega
 Begin
   R:=Encodetime(12,00,00,0);       // ** Coding for 12 Hours
   R:=1;                            // ** Coding for 24 Hours
   R:=Now - ((R)* 120);             // ** Coding for 120 days less
   PurgeHisto (Datetimetostr(R),'Backup.dbf'); // Purge the historical file
 End;
End;
```

Stop the historic recording task by program

```
If GetTask('PCHISTO')<>4 then  //* 0=Run 1=Stop 4=Absent
Begin
  RunTask('PCHISTO',False);    //* Stop Realtime task PCHISTO
End;
```

For re starting: `RunTask('PCHISTO',True);  //* Run Realtime task PCHISTO`

Exporter to excel

The **ExportHisto** function which is included in the **OBJHISTO.VPU** library allows exporting a historical variable on a specific date or from a date until the end of the historical file.

```
Function ExportHisto (HName,ExcelFileName : String; TDate : String; DateExclusive
: Boolean) : Boolean;
```

The parameter DataExclusive boolean type can be passed or non missing. Visual I/O will affect a state to be false to this missing parameter (optional parameters) automatically.

Example 1 Export of the historic of the variable ' W0 ' starting from the date specified in the character string LaDate.
All the historics of W0 starting from Ladate and higher will be exported into the FExcel.XLS file.

```
Var
  LaDate : String;
If button Then ExportHisto('W0','FExcel.XLS',LaDate); // Visual I/O affect a state
                                                      to be false to the missing parameter.
```

Example 2 Export of the historic of the variable ' W0 ' starting from the date specified in the character string LaDate. All the historics of W0 only at *Ladate* will be exported into the FExcel.XLS file.

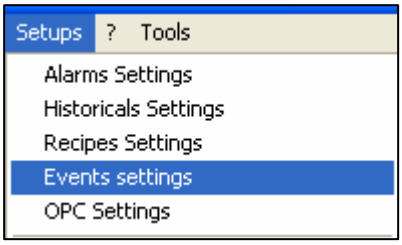
```
Var
  LaDate : String;
If button Then ExportHisto('W0','FExcel.XLS',LaDate,True); // Dateexclusive=True
```

Call Excel by program

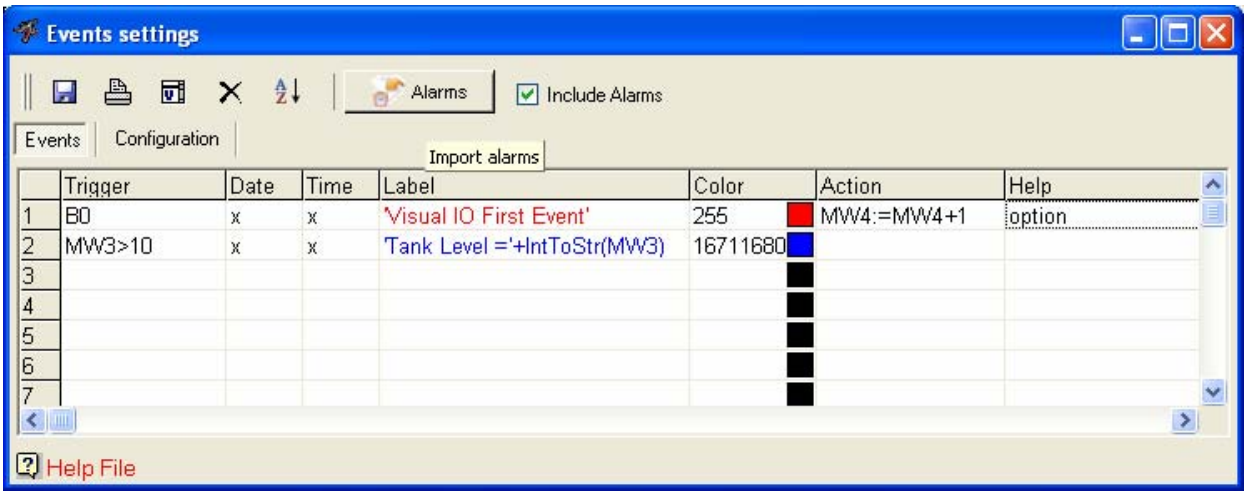
You can display a DBF file directly in Excel. **Shellexecute** launches Excel (open) prints (Print), the specified file. Here *Histo.dbf is* the file of the historics.

```
Uses Windows;
If Button Then
    ShellExecute(0, PChar('Open'), PChar('excel.exe'), Pchar('Histo.dbf'), nil,
                    SW_SHOWNORMAL);
```

Events management



A grid for setting the parameters on program Events. The events are assimilated to the alarm treatments. However, an event generates a simple or complex action. Events like alarms are displayed in run in a special component (listbox).
The result of these settings is saved in the file named PCEVENT.pas and PCEVENT.VPU (compiled).



	Trigger	Date	Time	Label	Color	Action	Help
1	B0	x	x	'Visual IO First Event'	255	MW4:=MW4+1	option
2	MW3>10	x	x	'Tank Level ='+IntToStr(MW3)	16711680		
3							
4							
5							
6							
7							

Alarms Button : To import the alarms into the events grid.
This function is done if you need to modify the text, colors .. of the alarms.
If you don't need to modify the alarm use the checkbox Include alarms.

Checkbox Include alarms : Add automatically the alarms at the end of the current list of events.

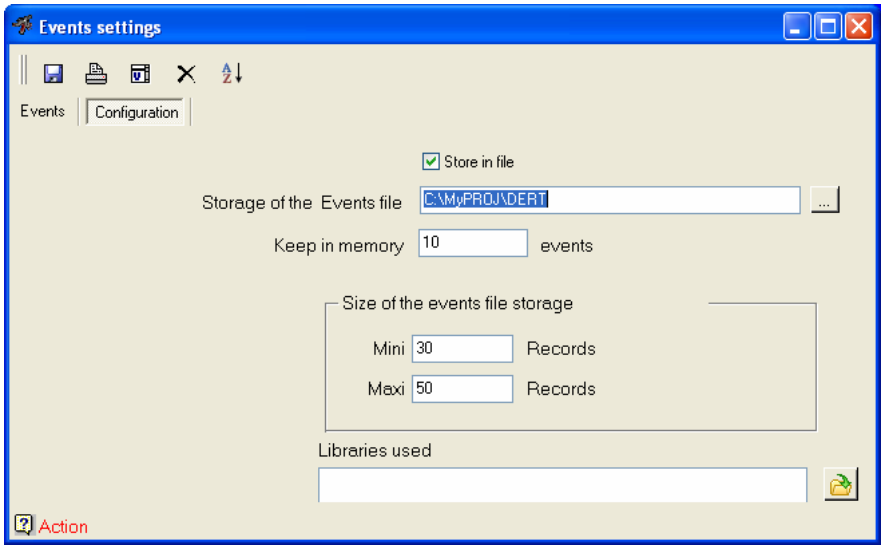
Trigger: Simple binary state (on rising edge) or complex expression to test. The result of this expression activates the event.
Date, Time: If x is written, the date or the time is inserted before the label.
Label : 1 / String **in quotes**
2 / concatenation of strings
Color : Display Text color in the list.
Action ; Optional. Little program is executed when the event is on (one time). This program is executed once on the rising edge of the event.
Help : Help Text (No treatment).

Note : If you want a automatic translation of the texts use the Translate function

Examples in

line 1 : Translate('Visual IO First Event')
Line 2 : Translate('Tank Level'=)+IntToStr(MW3)

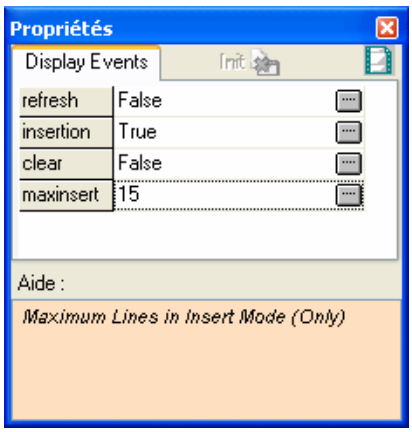
ConfigurationTab

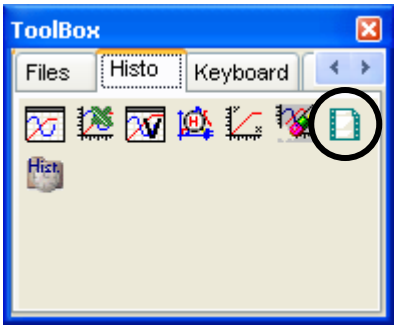


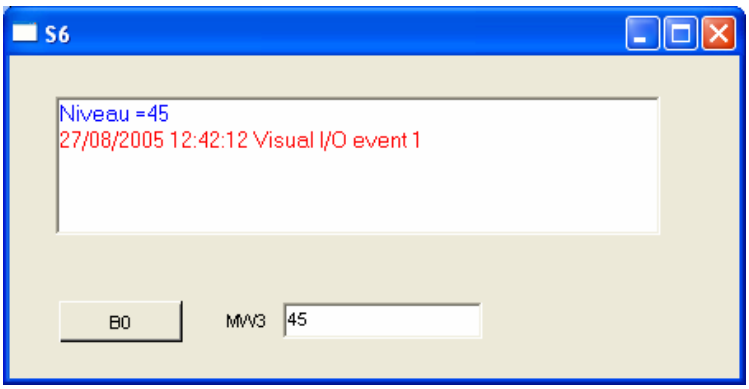
Store in file : If this checkbox is checked, the events are saved in the EVENT.TXT file. Otherwise, no save is done.
Storage of the events file: If no path is specified, the current directory is used. The events are stored in a file named 'EVENT.TXT directly readable by Excel.
Keep in memory: Define the memory buffer of the X last events.
Size of the events file storage (EVENT.TXT) defines the maxi and mini recorded events to be kept in the file.
Libraries used : Allow using external procedures and functions in the column *Label* and *Action*.
☞ **Note** : To understand the actions correctly, the source program generated is readable in PCEVENT.PAS.

Component for the management of the events

Display events







Refresh : To refresh the list of the events only when insertion is false.

Insertion : To define the display mode.

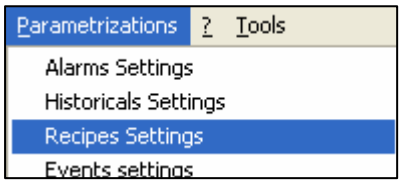
True : Only the events in real time are displayed. Maxinsert is the count maximum allowed.
If Maxinsert=-1, the list is filled without maximum restriction.
False : The events in real time are shown but the old events are also listed.
The X events kept in memory are always displayed. Memory is defined in the configuration tab.

Clear : Clear the list.

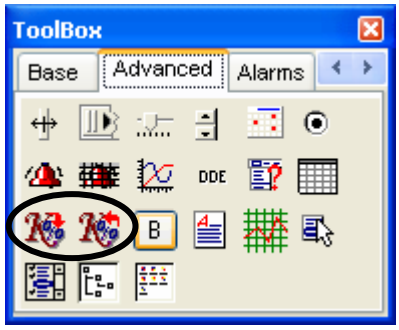
Maxinsert : Maximum visible lines is enable only if insertion is true

The events acts like historic. In other words, if the event disappears, the text stays displayed in the list.

Simple Recipes Management

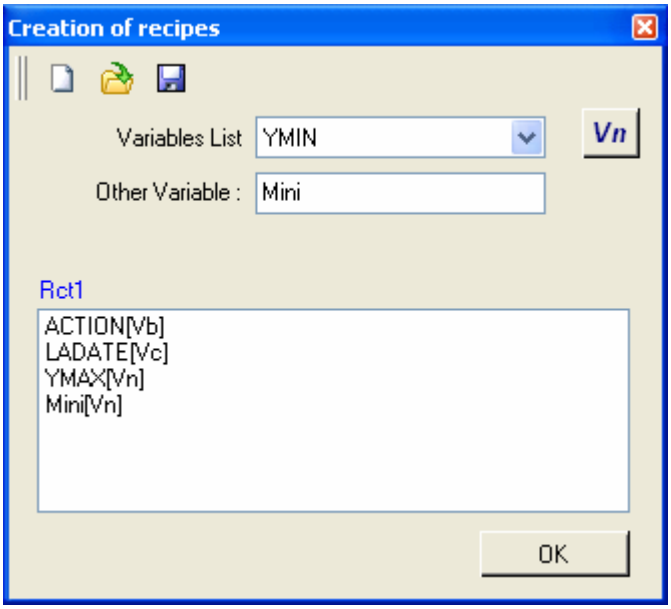


In the main menu
Click on Recipes settings.



Recipes configuration

Recipes define a list of variables affected by the loading of a **formula**.
The recipe configuration is only a declaration of the variables used by the formulas.

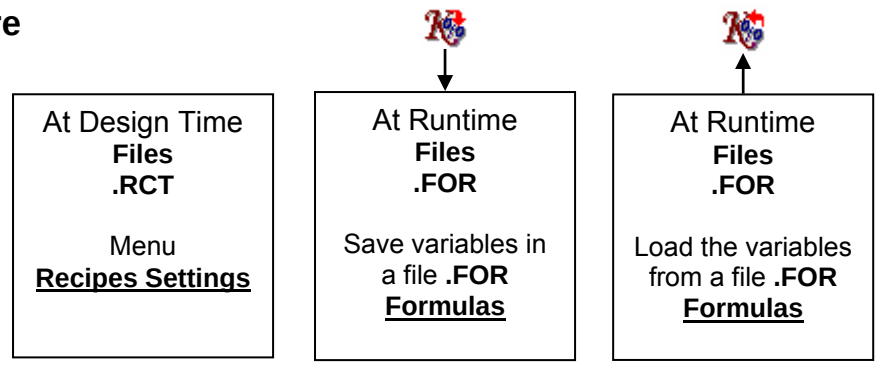


Here in the Rct1 recipe defined the ingredients to be used.

ACTION	Boolean type
LADATE	String type
YMAX	Numerical type
Mini	Numerical type

The component "*save a formula*" will save the values of these 4 ingredients in a file under a specified name.
Before the component "save formula" , the component "*load formula*" will download the variables with these 4 ingredients with the saved values.

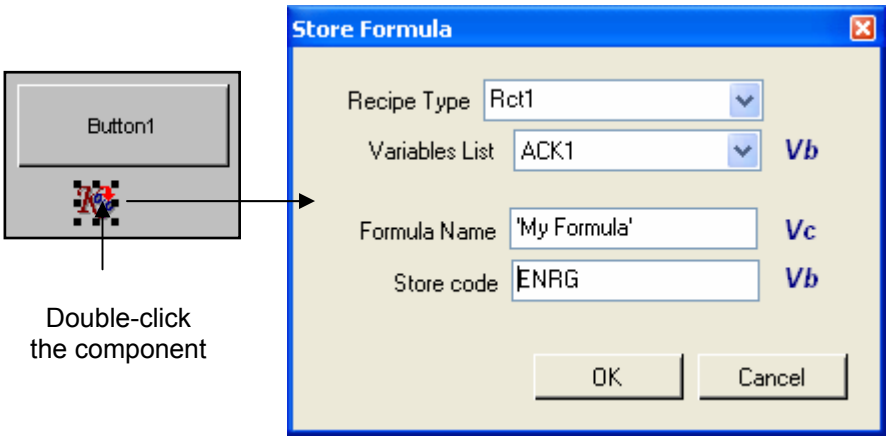
The structure



Recording a formula

This expert records values in a file: Bits or Strings.

To record a formula, you must know the ingredients that must record. Those are described in a recipe which has been done by the programmer with the help of the parameter recipes settings. (Parameterizations in the main menu of the Visual I/O editor).



Choose the **Type of recipe** on which the formula must operate.

Formule Name: Enter the name of the file which must be saved under the formula.
If this name is already known, this name must in quote (constant character string). If this name will be given only at runtime, then indicate the name of the String variable that will contain the final name of the file.

Store Code: Indicate a Boolean variable (on rising edge) to start the formula safeguard (recording on disc).

Details:
For example, if the recipe contains the variables Action, YMAX and Mini, this command will record the values of these 3 variables in the formulas file.

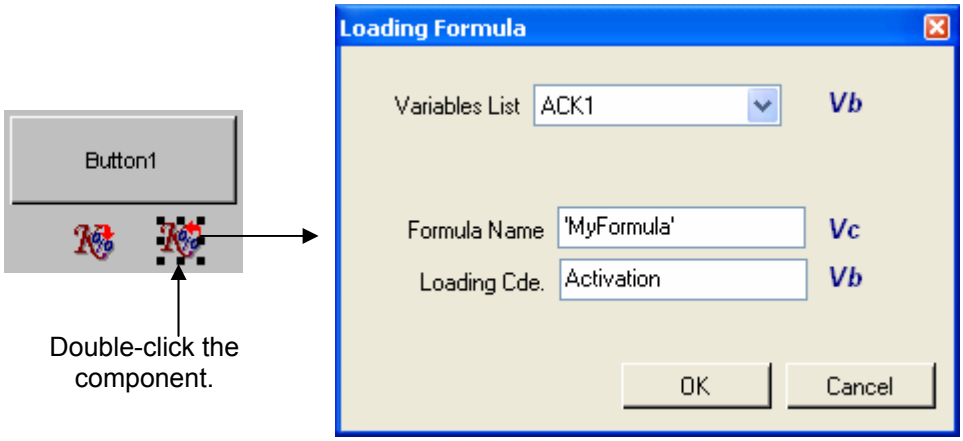
Generally, you do not know these values and recipes before the values constitute a recipe. It is necessary to determine them in process, to record and to restore them when a process changes.

☞ **Note**: The advantage of the recording and the loading of formulas lies in the fact that these treatments are carried out in basic spot, and that the form containing these objects can be invisible.

Bits can do activation; it is possible to associate a Grafcet with a sequence of the formulas loading.

Loading a formula

This expert downloads the ingredients (variables) of a recipe from a given formula file (file coming for disk).

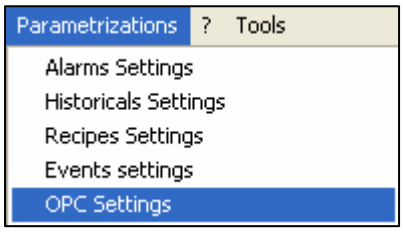


The formula name is specified in the '**Formula Name'** field.

The download activation for the variables constituting the recipe is done when the Boolean variable specified in '**Loading Cde'** goes True (on Rising Edge).

☞ **Note**: We do not need to specify the name of the recipe on which model this formula is formatted, because the name of this recipe is recorded in the formula file itself at the design time.

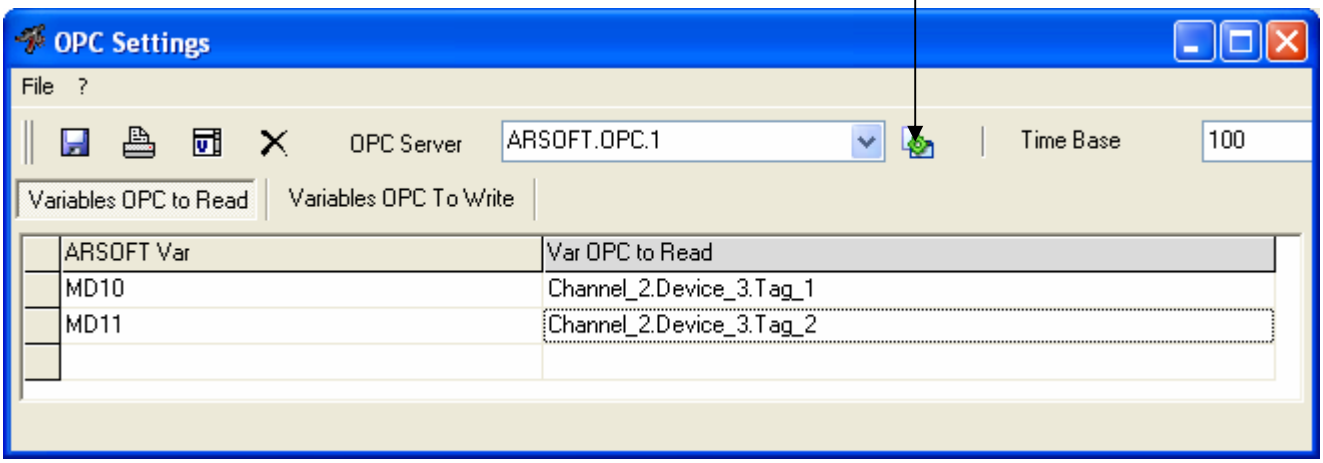
OPC Client management



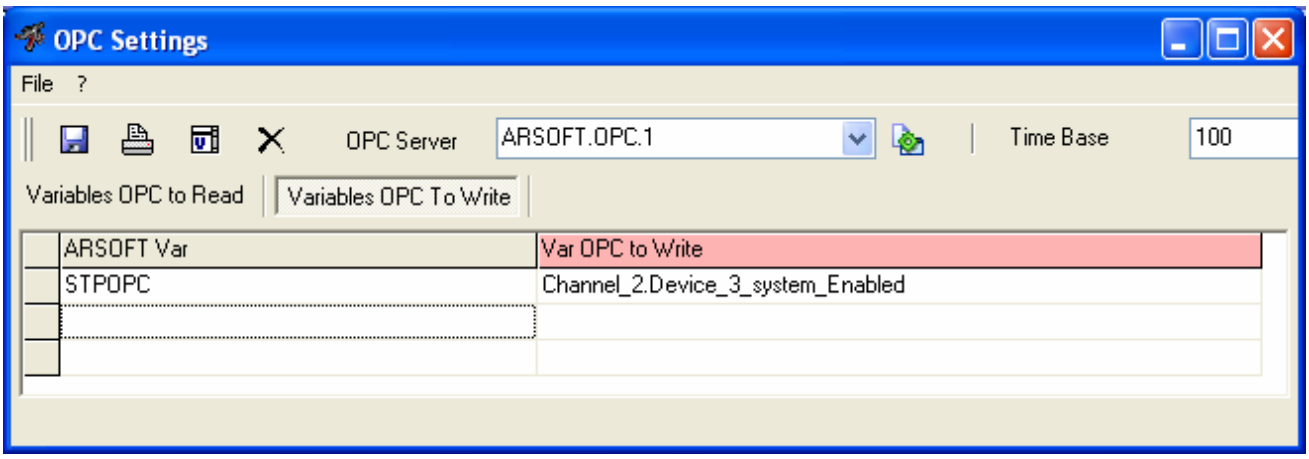
A grid of configuration is displayed to configure the OPC connections. The application generated will be an OPC client Application (OLE for Process Control). OPC defines standard objects, to access devices in an open and standard manner which is developing.

What the OPC servers the most are communication drivers to exchange data between PLC and scada applications.

After the OPC server installation, this one is registered in the register of Windows. To know its name and to be sure of its presence, this dialoxbox allow **scanning all the OPC servers registred** in your PC..



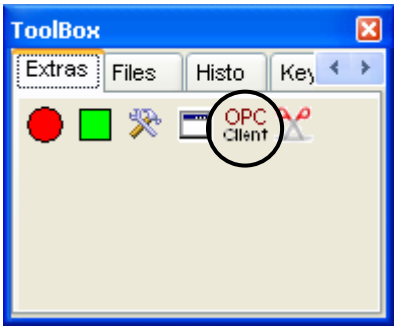
The OPC connection Reads and Writes variables in the OPC server. A tab for Reading values is available to affect Visual I/O variables with these of the server.
Time Base defines the frequency of the reading of the values from the server to Visual I/O.
If this value is 0 the reading is done every 10ms.
ARSOFT Var : Visual I/O variable receiving the value of the OPC server variable.
Var OPC to be Read: Name of the OPC variable to be read in the server.
This name is defined in the OPC server which you want to use (no conventions set).

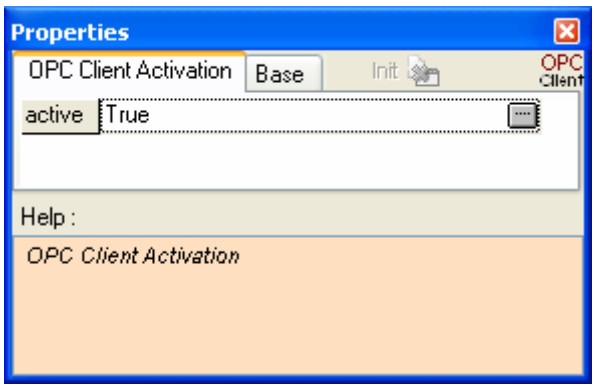


The Writing part is similar to the reading tab (2 columns). After being saved, a source file is generated and compiled automatically named **OPCCLIENT.PAS and OPCCLIENT.VPU**. The source file is readable, modifiable and can be compiled again. The OPCCLIENT.VPU program is added automatically to the list of the priority tasks in the final application (see menu Utilities/Application Constitution).

Activation of the OPC client connexion 

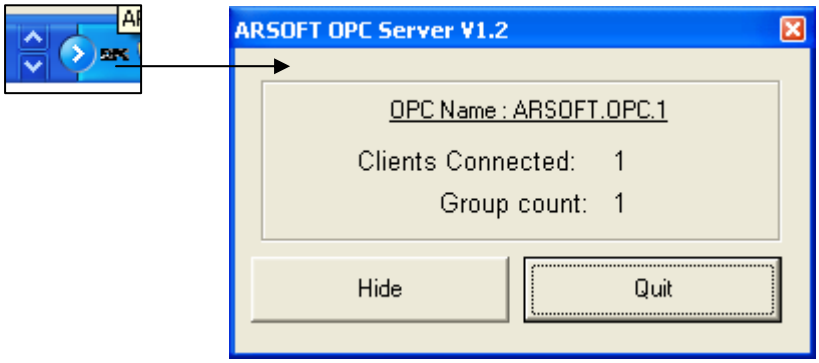
For reading and writing variables will be done in the OPC server, you must place in the main form this component (for a cyclically operation). Thanks to its property this component enables stop/run the OPC link..





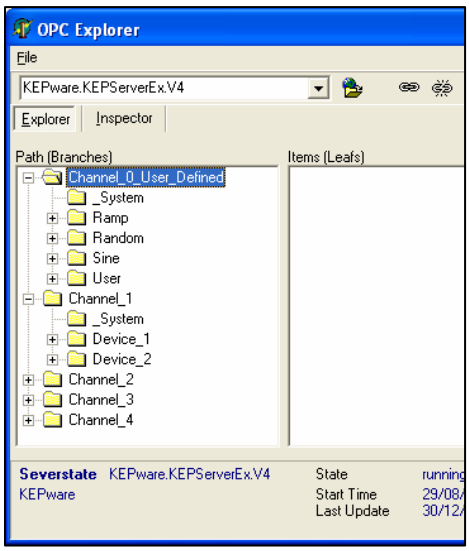
Transforming your application in an OPC Server

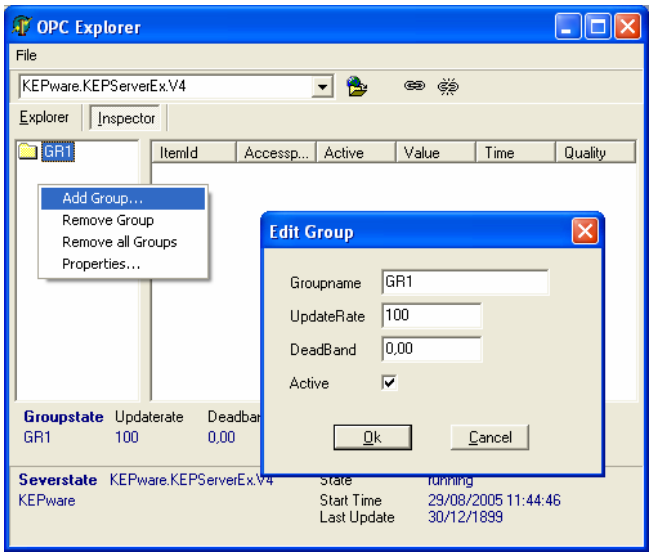
To transform your application into an OPC server, you must install the OPC server from your CD-ROM or download the OPC server on ARSOFT site (www.arsoft-int.com *resource, OPC Server for Visual I/O & Visual PLC applications*). The OPC server for Visual I/O is a simple executable running in the task bar of windows and gives an OPC interface to all the Global variables.
The server name is **ARSOFT.OPC.1** and the variables have the same names with the names which is declared in the global variables of your application (W0, MD10, B0, BP1, Tablo[1] etc.).



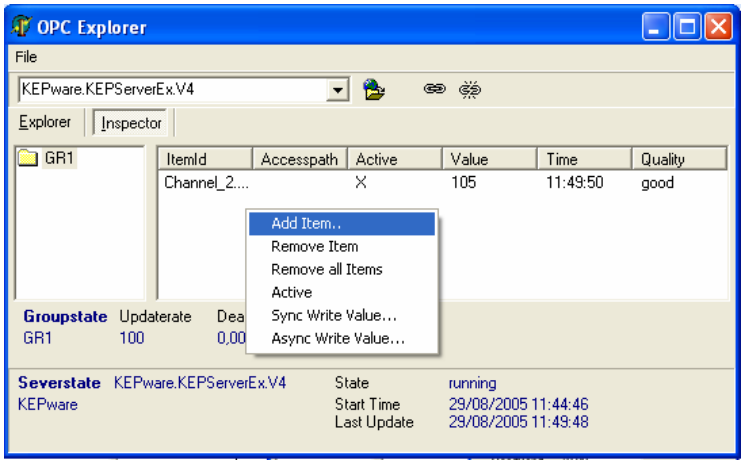
ARSOPC.EXE is a simple executable file, which will transform your application in OPC server. This server will be launched automatically when a client application asks for a Read or a Write command of a global variable. If the server is already launched, this one is visible as a little icon in the task bar of windows.

A utility (OPC Client) is delivered with the CD-ROM. This utility **dOPCExplorer.EXE** allows the connection to any serve rand to display the exchanged variables, read or write them.





This utility is used to know the list of the available variables. Before reading or writing in a server, you must select it in the upper ComboBox connected by one of the 2 buttons on top right. Then you must create a group including the variables to be read or written (OPC).



◄ Right click on the grid on the right for adding a variable to read or to write.

This pop up menu allows forcing the state of a variable.

☞ **Note**: This utility is free of charge.

The Report generator

Visual I/O allows printing reports. The report generator has modest functionalities but often be sufficient for the industrials applications. These reports can contain:

- Lines
- Rectangles
- Round rectangles
- Circles
- Ellipse
- Texts
- Values
- Bitmaps or Metafiles
- Frames
- Trends

To design a report, a form is used as a model. In the form used as framework for the report, we will be able to place graphic elements (Static or Dynamic). This synoptic must be included in your final application (see chapter Application Constitution).

Static elements:
These are graphic shapes in background such as lines, bitmap logo, circles etc...

Dynamic elements:
Dynamic elements are elements such as texts or values that will have an undefined value at the design time of the report. These dynamic components will receive their values just before the printing of the report; they are set with the respective variables at design time.

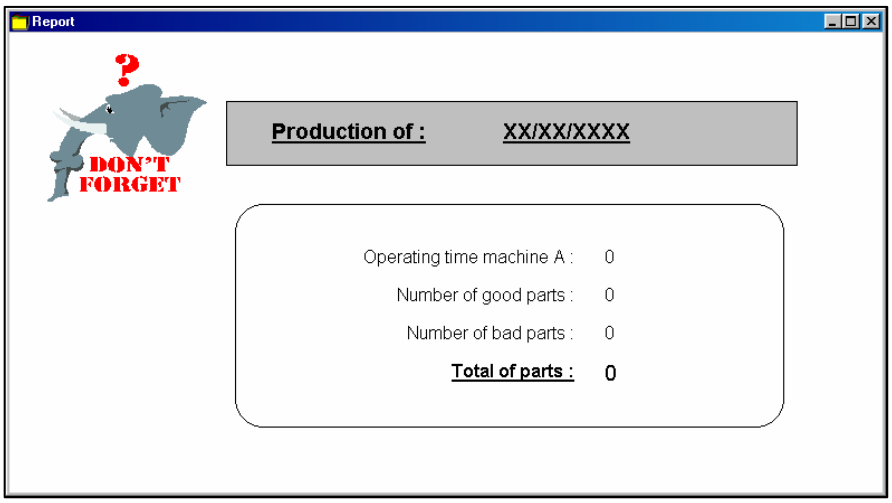
All the elements are active before printing. This allows hiding, displaying or moving by traditional instructions, especially printing your report according to your criteria.

The resolution of the printer on which you will print is higher than the form definition being used for the constitution of your report. If the synoptic is too small to place all the elements, to move a shape out of the screen, the scroll bars will appears.

Place the static elements of your report. Then make some printing tests to determine the limits of printing.

Example of report

On this form we have positioned some static elements (in background) :
A Metafile 'Don't Forget'., rectangles and Texts.
Dynamic elements:
A text XX/XX/XXXX that receives the date before printing. 4 numerical values without decimals which are affected by the expert to global variables.



If you decide to print this report by program, the elements drawn above will appear without modification, except for the numerical values that will be affected by theirs current values at printing time. If the name of this form is *MyReport*, it must be included into your final application to be used as a report (see menu Application Constitution).

The printing command for the report must be made only by program

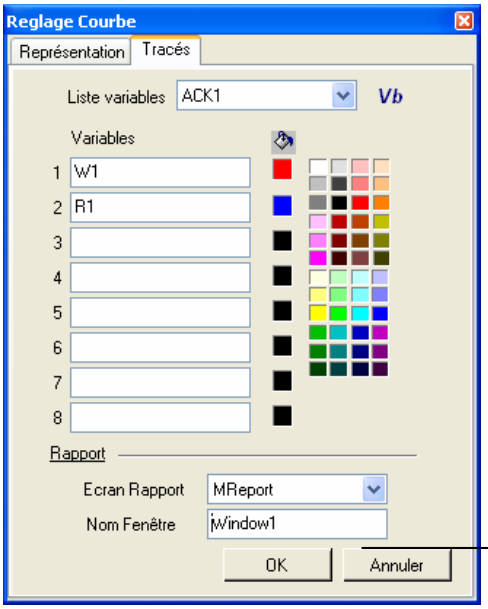
Example for printing a report:

```
 If Button Then PrintReport('MyReport') ;
 If Send   Then PrintReport('S8') ;
```

The report is printed on the printer (current printer in Windows).
You can make a **hardCopy** of the screen by calling the Function **HardCopy**;
To modify your printer in *'landscape'* , you can use the function **SETPRINTERFORMAT** (Portrait: Boolean);
If Portrait=True, the printing will print in portrait mode, otherwise in landscape.

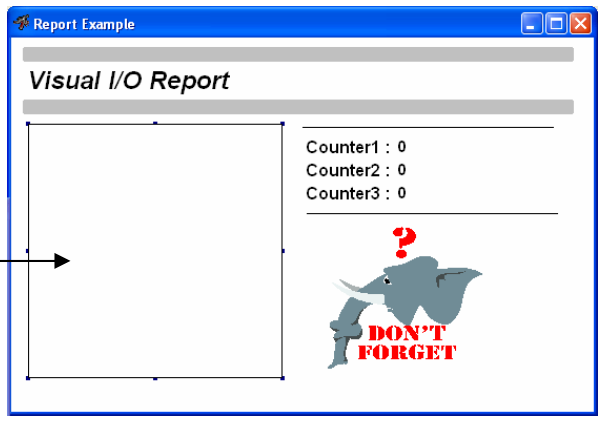
Including trends in a report

As seen before you can include a real time trend in your report (see chapter simple trend display).



The form below is normally done with the graphic editor, but will not be displayed to the screen but will act as a report framework.
This form will be printed and the window control will replaced by the trend at the execution time of the **PrintReport** procedure.

This form is named *MyReport* and the window control *Window1*.

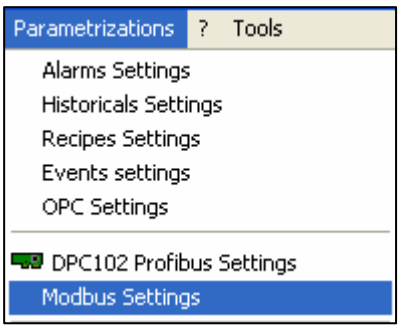


Incorporation of a trend in a report by program.

If you develop your own component to draw curves, it is possible to affect a curve to a window control in a report by program by using the procedure **TrendToReport**

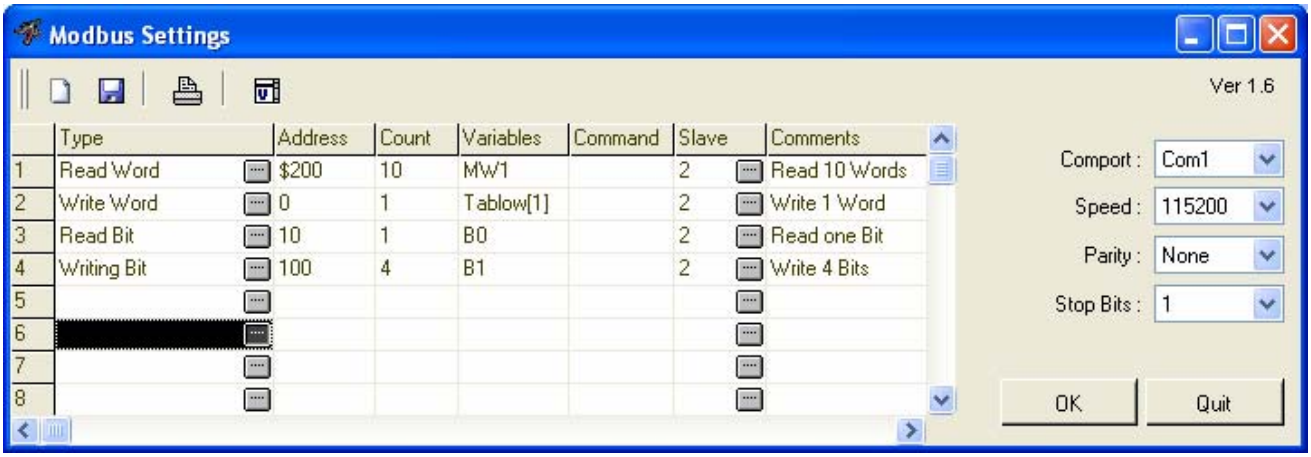
```
SObject Procedure Window1;
Var
   THDLE : Integer;
Begin
If Start_Visu Then
 Begin
   THDLE:=CustomTrend(10,5,1,1,0,450); // ** Create a trend framework
   TSetPencilColor(1,255);
   TSetPoint(1,10,10);
   TSetPoint(1,100,100);
   TSetPencilColor(2,1255);
   TSetPoint(2,20,20);
   TSetPoint(2,200,200);
   TrendToReport('Xreport','Window1',THDLE); // Affect this window to the report
 End;
End;
```

Modbus Configuration



This grid of parameters describe the recurrent communications. Those communications can read or write bits and words (16 bits) in Modbus slaves. Before describing the communications, you must declare a list of Words (WORD) and bits (BOOLEAN) receiving or reflecting the states variables in the slave equipment.

After validation by clicking on the button OK, the system generates the final Modbus compiled driver automatically.



☞ **Note**: *A right click with the mouse allows copying and duplicating lines.*

Example:
I want to read 10 words in a slave equipment number 2.
1/ Declaration of 10 Words in global variables.

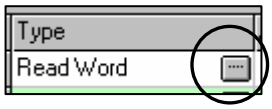
```
W1,W2,W3,W4,W5,W6,W7,W8,W9,W10 : Word ; Or
TabloW : Array [1..10] Of Word ;
```

The first line allows reading in the slave #2 of 10 consecutive words (16 bits) starting from the address 200 in Hexa. The values of these words will be stored in W1 to W10. The column Variables defines the first word for the reception or the emission. If several words are read, the first word will be stored into the specified variable then the following into the words consecutively declared in the global variables.
The column Command (optional) set a bit which will be reset by Modbus communication. The Bit must be set (True) to activate the concerned communication (Read/Write Bits or Words). If the dialog is ok, the system resets this bit (false = receipt acknowledgement).

The MODBUS Driver

After validation (button Ok), Visual I/O generates a compiled driver regarding of this configuration. This driver is implemented automatically in the current application (Menu: application constitution / Priority tasks). This driver will be executed in the priority task to perform a communication at maximum Speed.
The program name generated is PCMODBUS.VPU (source PCMODBUS.PAS is readable).

The available modbus commands are:

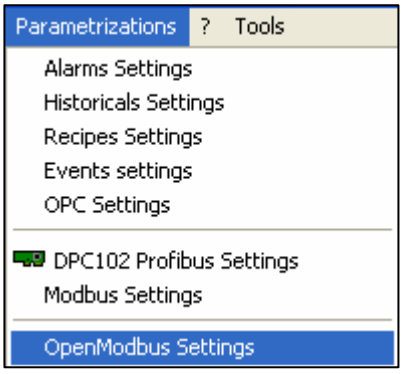


Read Word, Read Internal Word, Write Word, Read Bit, Read Internal Bit and Write Bit. To choose the dialog type, double-click on the small grey button in the cell.
It is possible to control the communication by the array of boolean
STATUSMODBUS : **ARRAY**[0..255] OF **Boolean**; *// declared in the global variables.*
A line of the grid corresponds at every indice of the array. If the line is correctly performed, the corresponding Boolean is set (True), otherwise false.

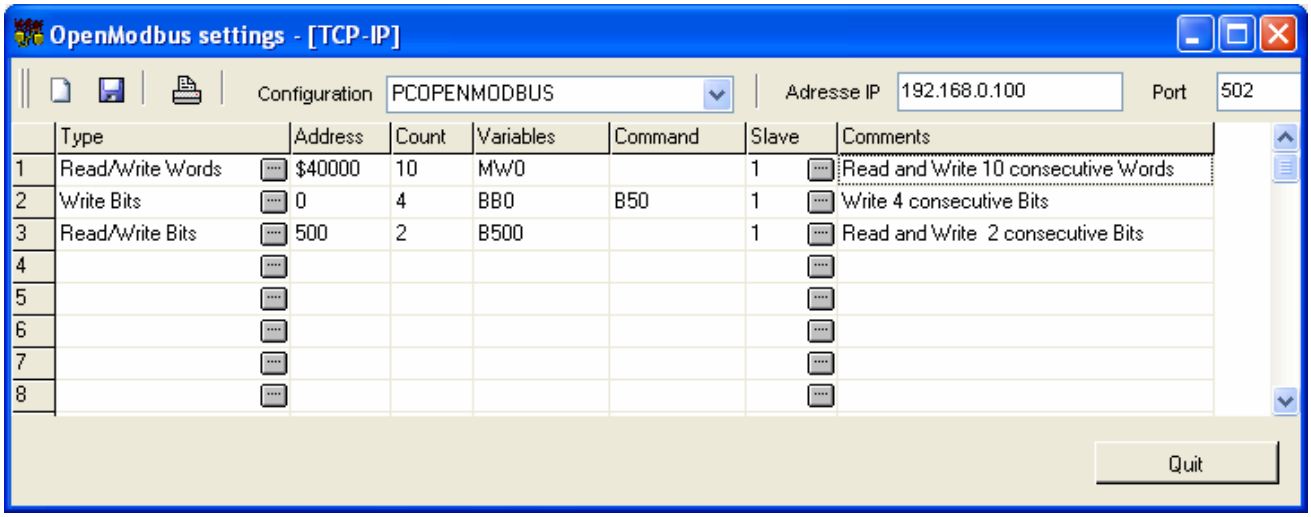
Cancelling the driver

Go to the menu utilities, Application constitution and priority task tab. Double click on **PCMODBUS** to remove it from the list.

OpenModbus configuration - Modbus Ethernet



This grid of parameters describes the recurrent communications. These communications can read or write bits and words (16 bits) in Modbus (TCP) slaves. This slave connects to the RJ45 connector of your PC (network connector).
Before describing the communications, you must declare a list of Words (WORD) and bits (BOOLEAN) receiving or reflecting the states of the variables in the slave device.
After validation by clicking on the button OK, the system generates automatically the final OpenModbus compiled driver named PCOPENMODBUS.VPU. The configuration file for the grid is stored in PCOpenModbus.oco.



	Type	Address	Count	Variables	Command	Slave	Comments
1	Read/Write Words	$40000	10	MW0		1	Read and Write 10 consecutive Words
2	Write Bits	0	4	BB0	B50	1	Write 4 consecutive Bits
3	Read/Write Bits	500	2	B500		1	Read and Write 2 consecutive Bits
4							
5							
6							
7							
8							

☞ **Note**: *A right click to copy/paste parameters lines.*

Configuration: Settings shown in the grid. You can create several configuration files according to various IP addresses. The port is normally set to **502**.
A shown VPU program corresponding to the file name is automatically generated and compiled. The program is inserted as a new driver into your final application (see the result in Menu: Utilities! Constitution of application).

IP address: Define the IP address for the OpenModbus connected to the slave. This address is defined in the slave itself.

Port : Normally 502 from OpenModbus specifications. Visual I/O allows changing this value if necessary.

The **Slave** Column sets a slave number but this value is not used because the slave address is given by the IP address.

The first line reads10 words in the slave from the address $40000 (40000 in hexa) and stores the result in Visual I/O words beginning from MW0 and consecutively. If one of these words is changed in the application, a write command is performed to the slave.

The second line allows writing in the slave 4 consecutive bits from address 0 with the values of the bits BB0 and so on. This writing is done when the bit B50 is True. When the writing is done by the driver, the command bit is reset (false) automatically by the driver, to perform an acknowledgement.

The third line allows reading 2 bits beginning at the address 500 in the PLC. The return values of these 2 Bits will be stored in the bit B500 and other word declared consecutively after. If one of the 2 bits changes in the program, a write command is performed for updating the slave connected.

The column Command (optional) affects a bit to activate the Modbus frame.
This Bit must be set (True) to activate the concerned frame (Read/Write bits or Words).
If the communication is right, the driver reset this bit automatically to perform an acknowledgement.

The OPENMODBUS driver (PCOpenModbus.VPU)

After clicking on the button Save, Visual I/O generates a compiled driver in regard to the parameters in the grid. This driver is implemented automatically in your application (Menu: application constitution / Priority tasks). This driver will be executed in priority task in your application to insure a dialog at the maximum speed. The program name for the driver generated is PCOPENMODBUS.VPU (source is PCOPENMODBUS.PAS is readable).

The available variables with OpenModbus

When you add an OpenModbus communication to your application, the system adds the following Global Variables automatically:

StatusOPENMODBUS : Array [0..400] Of Boolean; is an array of Boolean (bits) indicating the number of the line in progress. In other words, a line in the grid is corresponding to each indices of this array.

ComOPENMODBUSOK : Boolean; this Boolean indicates that the windows' socket (IP Address + port) is open. This Boolean does not indicate that the OpenModbus Slave (I/O, PLC) is physically present.

Note: If the communication is break down (unplugged Cable, device off), rearm procedure is automatically done.

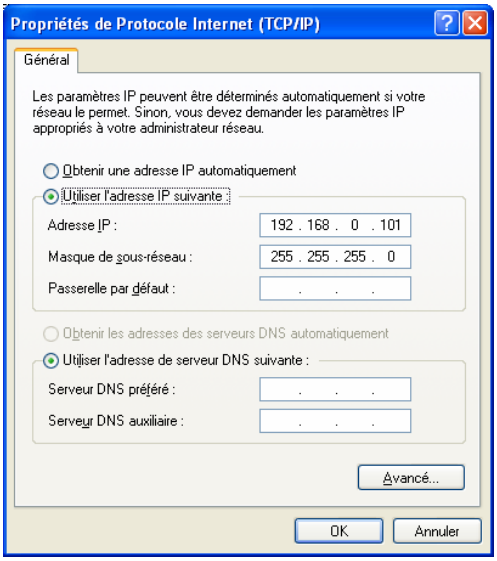
The variables are used in the driver (OpenModbus.vpu).

OpenModbusCounter: Integer; Communications count.

StopOpenModbus : Boolean=False; This Boolean to Suspend/Resume the communications with the slave device by program.

OpenModbusDisconnected : Boolean=False; If this Boolean is Set (True), means that at least a communication OpenModbus has be done and the slave does not reply (disconnection, power supply off..).

Set the ip address in your PC (Network connection)



If the Slave is connected to the following IP address 192.168.0.100

You must set your Ethernet card in your PC like that with address 192.168.0.100 subnet mask 255.255.255.0

The usual subnet mask is 255.255.255.0

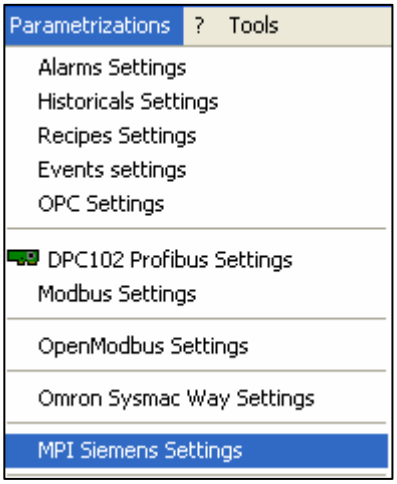
A **subnet** allows the flow of network traffic between hosts to be segregated based on a network configuration. By organizing hosts into logical groups, subnetting can improve the security and performance of network.

Types of communications:

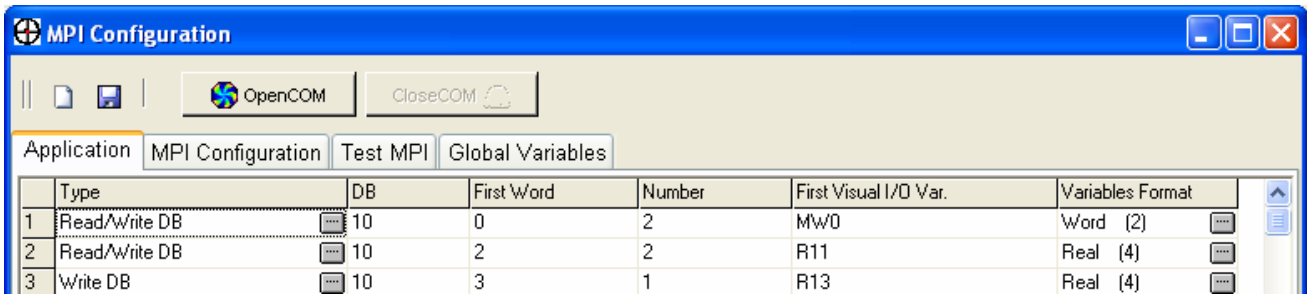
- Read internal Words : From Device to PC
- Write Words : From PC to Device
- Read internal Bits : From Device to PC
- Read / Write Bits; : From Device to PC and From PC to Device (if changes).
- Read / Write Words : From Device to PC and From PC to Device (if changes).
- Write 1 Bit : From PC to Device
- Write Bits : From PC to Device
- Write 1 Word : From PC to Device
- Read Status : From Device to PC

Note: If you need to read Reals in a device, you must declare **Single** variables in your global variables and use Read/Write words command with 2 words for one single read. Single are the Intel Format.

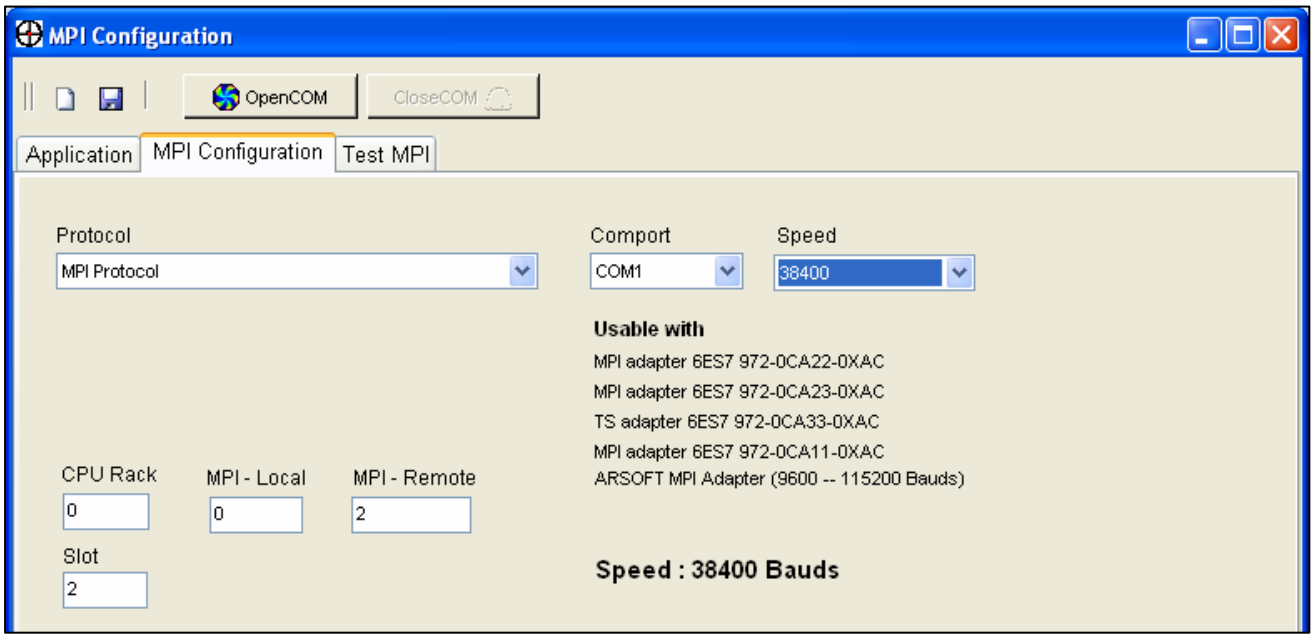
MPI Configuration – Serial and Ethernet



This grid of parameters describes the recurrent communications.
These communications can read or write bits and words, Integers, Reals under MPI protocol. The PLC is connected through the RJ45 connector of your PC, by the serial link, or by USB via a MPI adapter.
Before describing the communications, you must declare a list of Words, integer Reals and Bits receiving or reflecting the variables states in the PLC (Global variables).
After save command, the system generate **PCMPI.VPU** which is the final DRIVER used in your application.
The grid is saved in a file file named PCMPI.CFG. The configurator is an external program to the Visual I/O editor. This configurator is written in Visual/O (sources available on request).



The first tab (Application) configures reading and writing to perform.
The Writing is performed on changing states, which allows optimizing the communication with the PLC.
It is possible to write or read DB, DI, Counters and temporizations. Other variables use the procedures and functions of the MPISerial.VPU library. The variables formats are specified by the ComboBox in the "*Variables Format*" column. It is possible to use words, integers or reals in the same DB. You need to set few lines. The **Read/Write** commands allow reading recurrently and also writing at the same address in the PLC. The write is performed only when the global variables change in your application. A correspondence between tables (PLC/SCADA) is directly possible.



The serial link is used with MPI/Adapters siemens or Vipa type (ARSOFT is provider too).
Different speeds depend on the adapter. The ideal speed is **38400 Bauds**.
Else the Ethernet connexion of your PC can be used directly with the PLC CPU or through an **IBH/Hilscher** adapter.

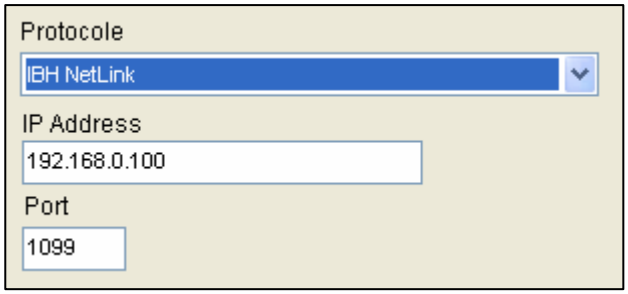
For the **USB adapters (**Siemens and compatible), choose **USB- PCAdapter**. You must install the CD Rom provided with the USB adapter to make possible using the S7Onlinx.DLL.

The fields you must fill are

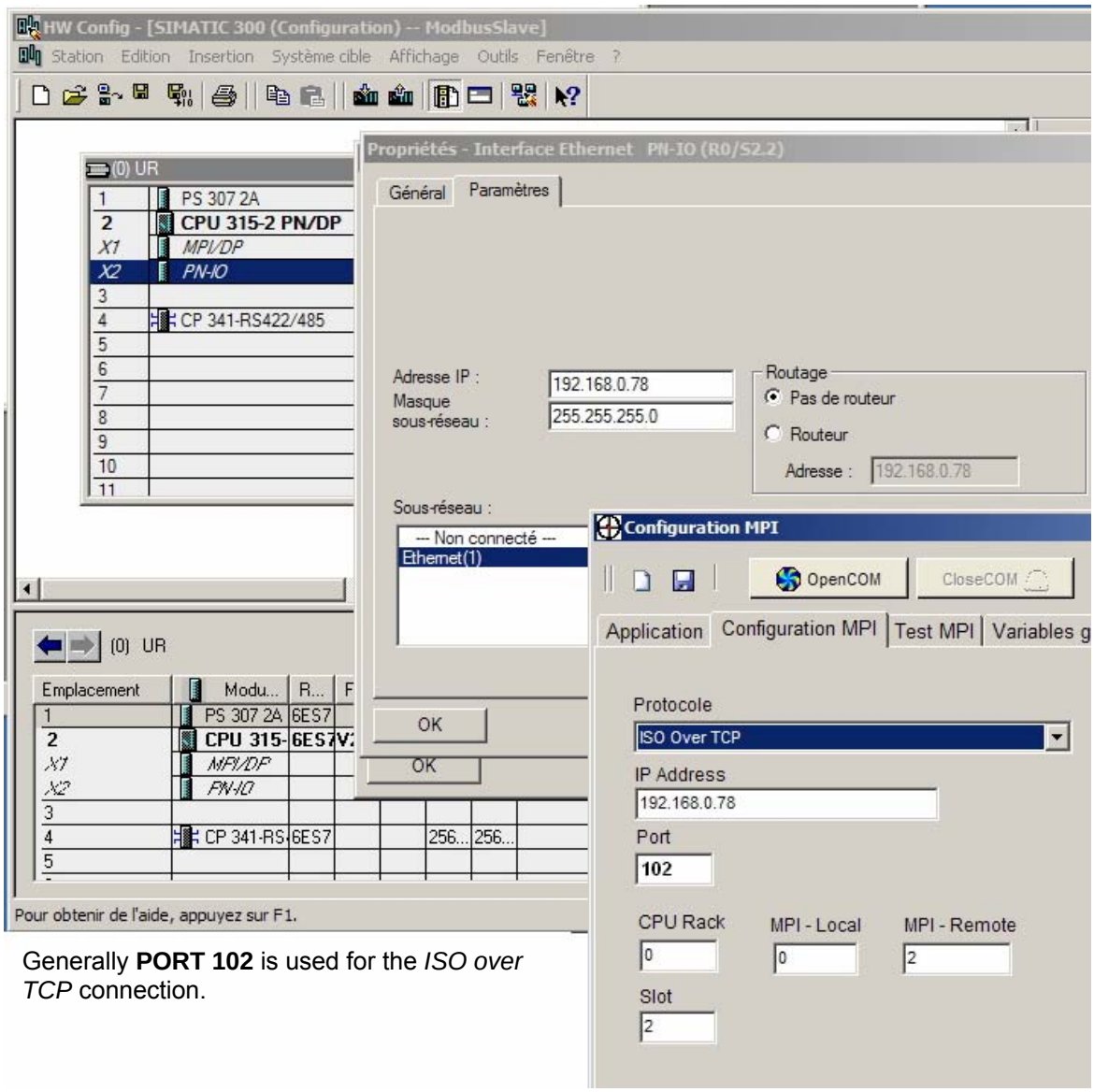
CPU Rack : Rack number (PLC Side).
MPI-LOCAL : MPI Number PC side (Generally 0).
MPI-REMOTE : MPI Number PLC Side.
SLOT : Slot number in the PLC.

Concerning the PLC side, it refers to the siemens specifications.
If the connection is done in TCP/IP through an adapter or directly connected to the CPU, you need to set other parameters:

IP address : Is the IP address where is connected Your MPI adapter or the PLC CPU.
Port : Is port number used in your PC for communication in TCP.
These 2 supplementary parameters are not specific to the MPI Connection. They are used in each TCP/IP connection.

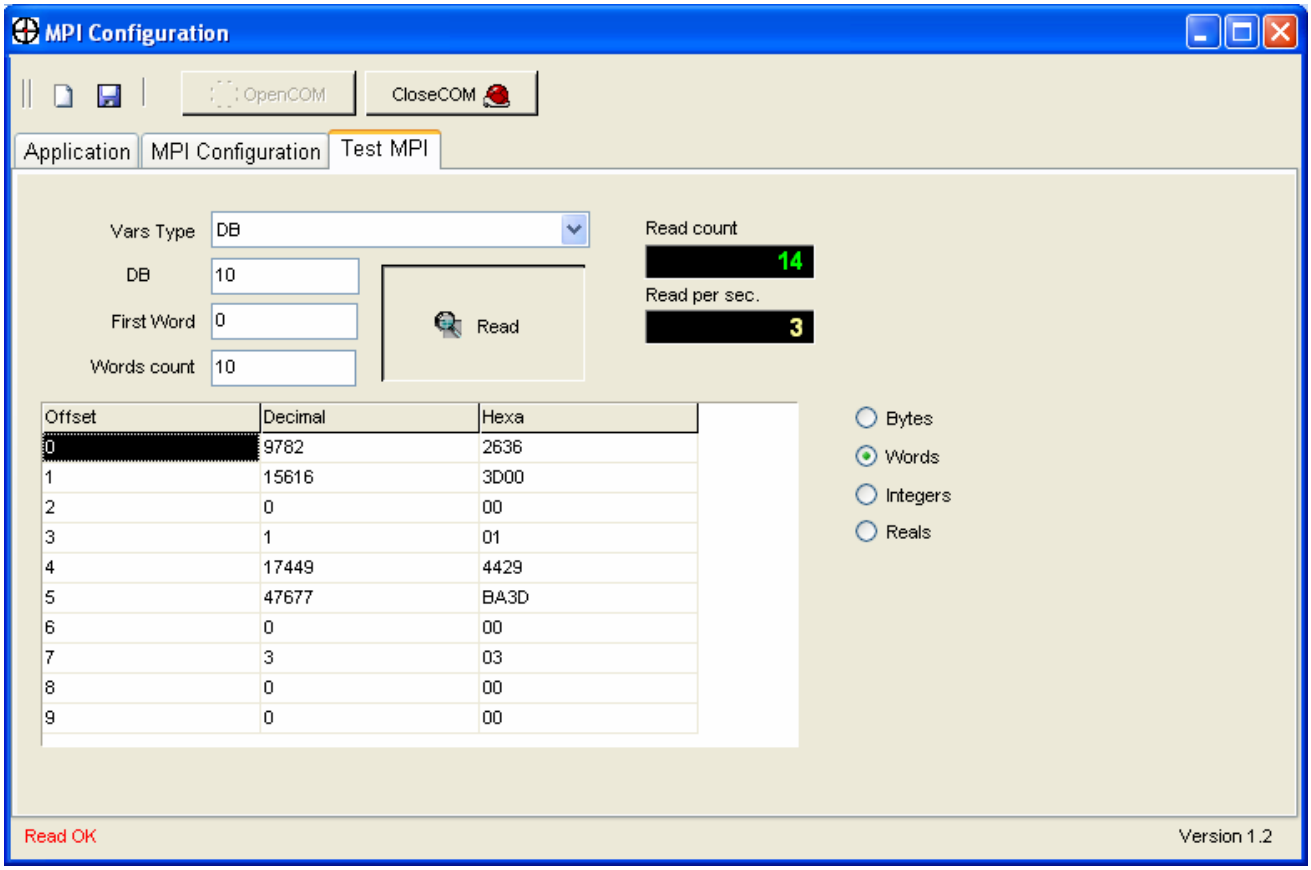


Communication with Siemens PLC Ethernet card in cpu type CPU 315 PN/DP



Generally **PORT 102** is used for the *ISO over TCP* connection.

The Last TAB is for the MPI test; allow testing the connection directly in the configurator like the Profibus configurator. The test holds the type of MPI adapter configured (serial, Ethernet).



1/ Click on the button OpenCOM to activate the connection with the adapter. If the connection is on, this button is grayed, the CloseCom button is enabled and the reading occurs. Choose the variable type to read in the PLC and its format and the number to read. The read values are displayed in the grid. It is possible to modify a value in the PLC. You can place the cursor in the column Decimal, and then hit a new numerical value in the selected cell. A frame to write is sent automatically to the PLC. If the communication breaks down, click on the button CloseCom then OpenCom again to reinitialize the whole communication with the PLC.

The PCMPI.VPU driver generated

After saving and compiling, the configurator set this driver automatically in the priority tasks of your application. It also adds supplementary variables to the global variables list of your application.

COMMPIOK : `Boolean; Returns` if the initialization of the communication is done:
1/ Opening of the serial communication port or Ethernet
2/ Initialization with the MPI adapter.
3/ Connection to the PLC.

This bit is reset if the connection with the PLC falls in timeout. If Visual I/O sends a frame and there is no reply coming from the PLC.

MPILASTERROR : `String;` this string returns the last error. If there is no error, this variable is empty.

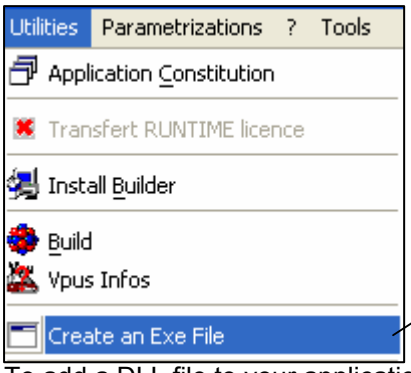
REARMMPI : `Boolean`; No more action Boolean is kept for compatibility. Because rearming MPI communication is automatic now (for example, in the case of timeout of the PLC).

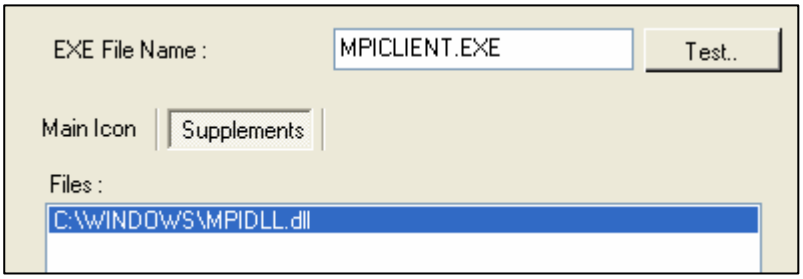
Procedure ReconnectMPI: is also available in PCMPI.VPU interface.

The source PCMPI generated is available in the Programs of Visual I/O editor.

Important :
To make an MPI communication, this PC needs the **MPIDLL.DLL** in the directory C:\Windows. Normally the MPI configurator confMPI.EXE delivery installs this DLL file.

If you deliver an executable, do not forget to include **MPIDLL.DLL.**





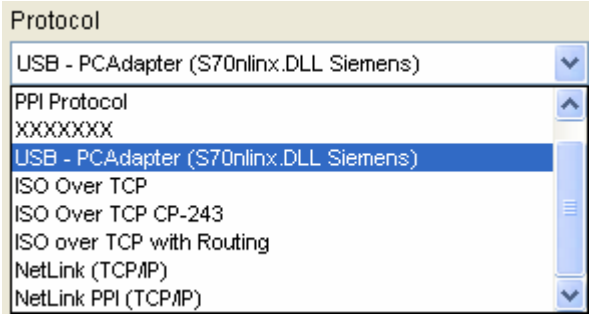
To add a DLL file to your application, it is allowed at launching time to send this DLL in windows' directory (C:\Windows) for the application which can use it.

The PCMPI.vpu driver uses an independent thread in your application to prevent blocking the rest of the application in case of time out with the PLC. By all means, your application will remain fluid.

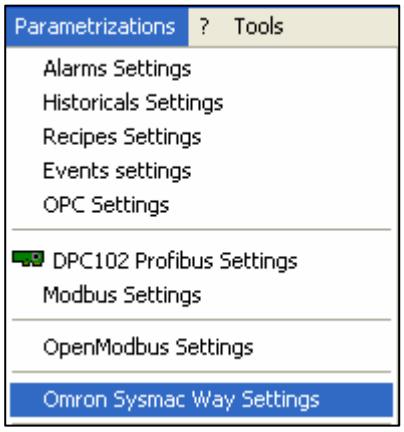
The possible different connections:
PPI Protocol (serial) PLC serie 200.
MPI Protocol
ISO over TCP
IBH Netlink
USB – PC adapter

Are 4 connections often use.

USB – PC adapter you need install the cd-rom from siemens.



Omron configuration



This grid of parameters describes the recurrent communications. These communications can read or write bits and words in a PLC OMRON under Sysmac Way protocol. Under this protocol only the Word type (16 Bits) are allowed. Before describing the communications, you must declare a list of Words and Bits receiving or reflecting the variables states in the PLC

After saving the configurator, a driver named PCOMRON.VPU generates.

<u>The available commands Omron are:</u>

Read Word(DM), Write Word(DM), Read Word(IR) , Write Word (IR), Read Word(HR), Write Word (HR), Read Word (TC) ,Read Bit State(TC).
It is possible to read words up to 30 or 120 consecutives bits per command line in a configuration grid.

The terms (DM), (IR), (HR) and (TC) define memory mapping whose structure is describeed below:

Zone of datas		Words	Bits	
IR	Zone Input	IR000 to IR009 (10 channels)	IR00000 to IR00915 (160 bits)	
	Zone Output	IR010 to IR 019 (10 channels)	IR01000 to IR 01915 (160 bits)	
	Zone internal	IR 200 to IR 231 (32 channels)	IR20000 to IR 23115 (512 bits)	
SR	Flags	SR232 to SR255 (24 channels)	SR23200 to SR25507 (384 bits)	
HR		HR00 à HR19 (20 channels)	HR0000 to HR1915 (320 bits)	
AR			AR0000 to AR1515 (256 bits)	
LR	Data link		LR0000 to LR1515 (256 bits)	
Tempo /Counter		TC000 to TC127 (Number of temporizations and counters)		
DM	Read/Write	DM0000 to DM0999 DM1022 to DM1023 (1002 words)		
	Errors	DM1000 to DM1021 (22 words)		
	Read only	DM6144 to DM6599 (456 words)		
	Debug PLC	DM6600 to DM6655 (56 words)		

The PCOMRON.VPU driver generated

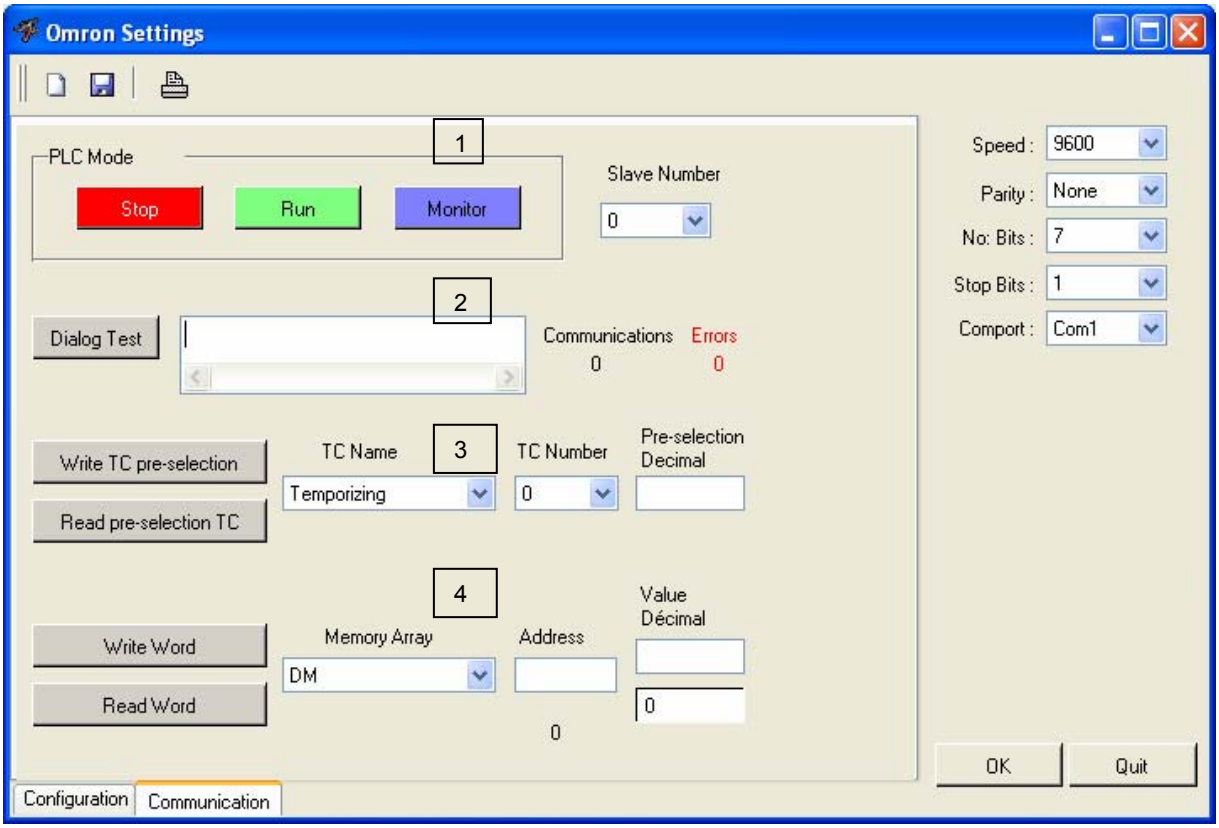
After saving and compiling, the configurator writes this driver automatically in the priority tasks of your application. It also adds supplementary variables to the global variables list of your application.

STATUSOMRON: ARRAY[0..255] of Boolean;
(True if the communication is good or in progress, False if communication is in error).

COMOMRONOK : Boolean;
This variable returns the result of the opening of the com Port (True if the port is open, False if not open).

A second tab proposes a dialog interface directly in the editor with a connected PLC.

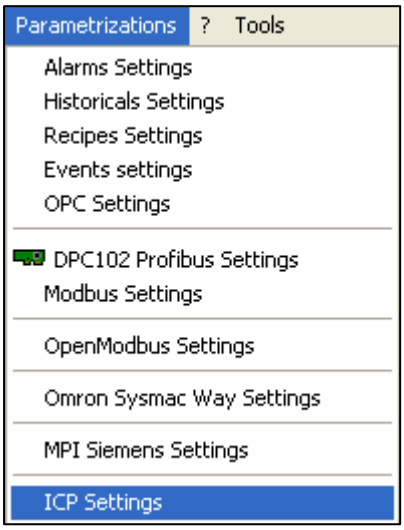
Test on serial communication



<u>The features available are:</u>

1 : Modification of PLC Mode. The Monitor mode allows actions at runtime.
2 : Test and diagnosis of the communication with the PLC.
3 : Reading and Writing of TC preset (Temporization or counter). The value to be edited and the returned value are in decimal on 4 digits.
4 : Reading and Writing in the memory zone IR, SR, HR, AR, LR, DM : The value to be edited and the returned value are in decimal.
Read the current value zone TC (Temporization or Counter): The value to be edited and the returned value are in decimal.
Reading word state of TC (Temporization or Counter) : The value to edit and the returned value is in binary. You can read it from right to left and show the output state of 4 consecutives TC.

ICPDAS Configuration



This grid of parameters describes the recurrent communications. These communications can read or write bits and words in ICP modules (and *Advantech* compatibles)
Before describing the communications, you must declare a list of Words and Bits receiving or reflecting the variables states in the slave modules.

After saving visual I/O generate the final driver **PCARSICP.VPU**.
This driver uses 2 external DLL **I7000.dll** and **UART.dll**.
These libraries (DLL) are located in the Windows directory or Windows\System32.

For testing in the configurator, you must only have the DLL UART.DLL.

ICP Modules Settings

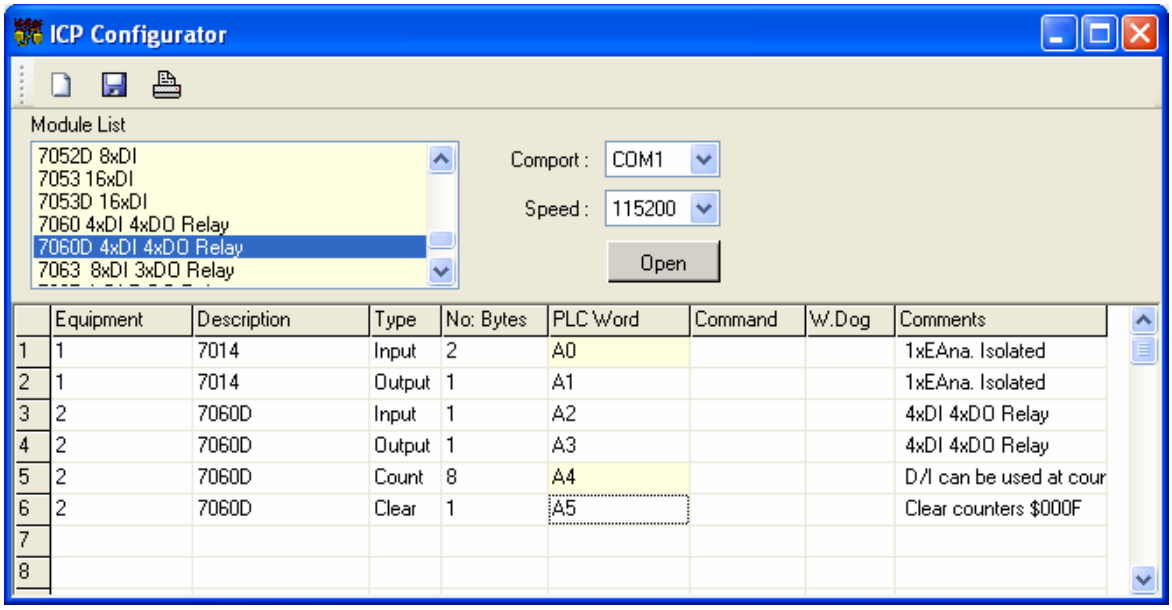
The configuration of the modules is done by the utility 7000util.exe (on your CD-Rom or on icpdas.com web site). This utility is enabled to modify the address of the module, its speed and its internal functionality (4-20 mA, +10 –10V etc for analog modules).
You can only modify one module at a time, follow these instructions:

- Switch off the power supply of the modules
- Set the micro switch behind the module to INIT position (for the recent modules) OR make a shunt between the GND and INIT* on the connector.
- Switch on the power supply (24V)
- Launch the **7000util.exe** now and follow (*your felling or the help file*)

Modules Configuration

Get the module by drag & drop from the top list box to the grid. This module is placed at the end of the list in the grid displaying the number of bytes needed in reception and in transmission.

After you must place the variable name in the corresponding column.



If few words are necessary, place only the first word's name. The rest will be stored in the consecutives words declared behind this global variable.

ATTENTION:

The **INTEGER** type is needed for exchanges with a Digital module.
The **REAL** type is needed for exchanges with Analog modules.

Modify the slave number if necessary.
Command : If no bit is specified, the polling is done cyclically by the program. If a bit is specified, the exchange with the module is done only when this bit is True, when the exchange is performed correctly the server reset this bit (pseudo acknowledgement).

Diagnostic: When you save, the program creates supplementary global variables for you automatically.

> **COMICPOK**: Boolean -> If comport is opened correctly
> COMICPOK is True if the comport is opened correctly
> COMICPOK is False error when opening the comport.
>
> **STATUSICP**: Array[0..255] of Boolean -> displays the state of each communication (corresponding to each module set).
>
> STATUSICP[5] is True if exchange with the module number 5 is ok.
> STATUSICP[5] is False if exchange with the module number 5 is wrong.

Using the counters:

The digital which inputs on certain modules can be used as counters (100 Hz).

Modules 7041D, 7052D, 7053D, 7044D, 7050D, 7060D.

When you declare these modules in the grid, 2 lines appears in supplement.

The first line for the counter (indicate here only the first word)
The second line allows a reset of the counters.

Each counter has associated a bit to reset its value (First Counter->bit0..counter4->bit3).
In the grid above if A5 is 15, the four counters will be reset.
If A5 is 9, the counter #1 and the counter #4 will be reset.

Watchdog (optional)

It is possible to affect a value to the watchdog of each module. The functionality of this watch dog is to reset the outputs of the concerning module physically when this module is disconnected from the master greater to a laps of time defined in multiple of 100 ms.

Attention to calibrating correctly the watch dog regarding to the speed used and the number of slaves connected on the network . The more the number of slaves is, the higher the general scrutation time is.

PLC Word	Command	W.Dog	Comments
A0			1xEAna. Isolated
A1			1xEAna. Isolated
A2		10	4xDI 4xDO Relay
A3		10	4xDI 4xDO Relay
W1	B1	10	D/I can be used at cour
Reset	B2	10	Clear counters $000F
		x100ms	

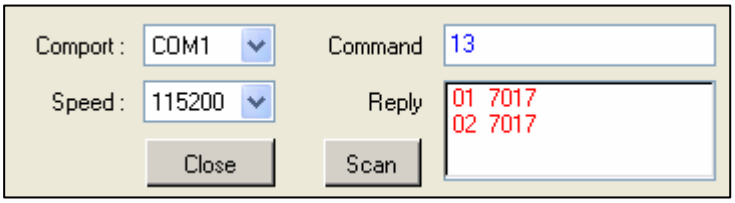
Watchdog for 1 second

Each ICP module has a front led. This led is on when the module is read cyclically by the master.
It is blinking when the module is disconnected from a certain time superior to the value of the watchdog.

Diagnostic Led of the module.

Line of command and Scan

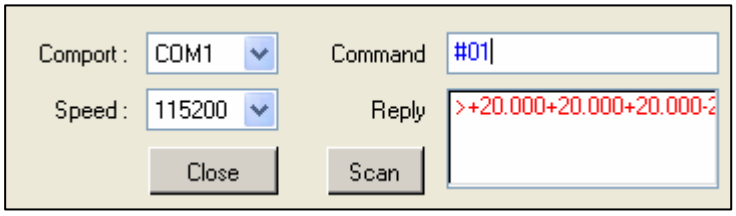
The ICP configurator is enabled to scan on the serial link, the whole connected modules under the communication speed is specified.



Here 2 equipments 7017 (1 and 2) are connected at 115 200 Bauds.

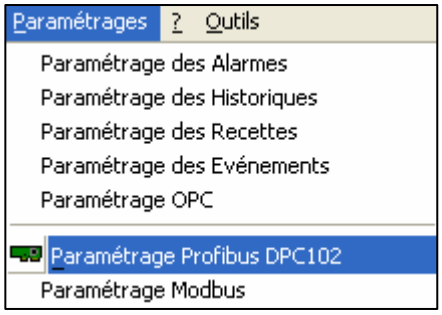
It is also possible to execute different lines of command (see modules manual) for each ICP module. These commands are to set the modules parameters, to read the digital or analog inputs etc..

Example: Read analog inputs from the equipment 7017 n°1.



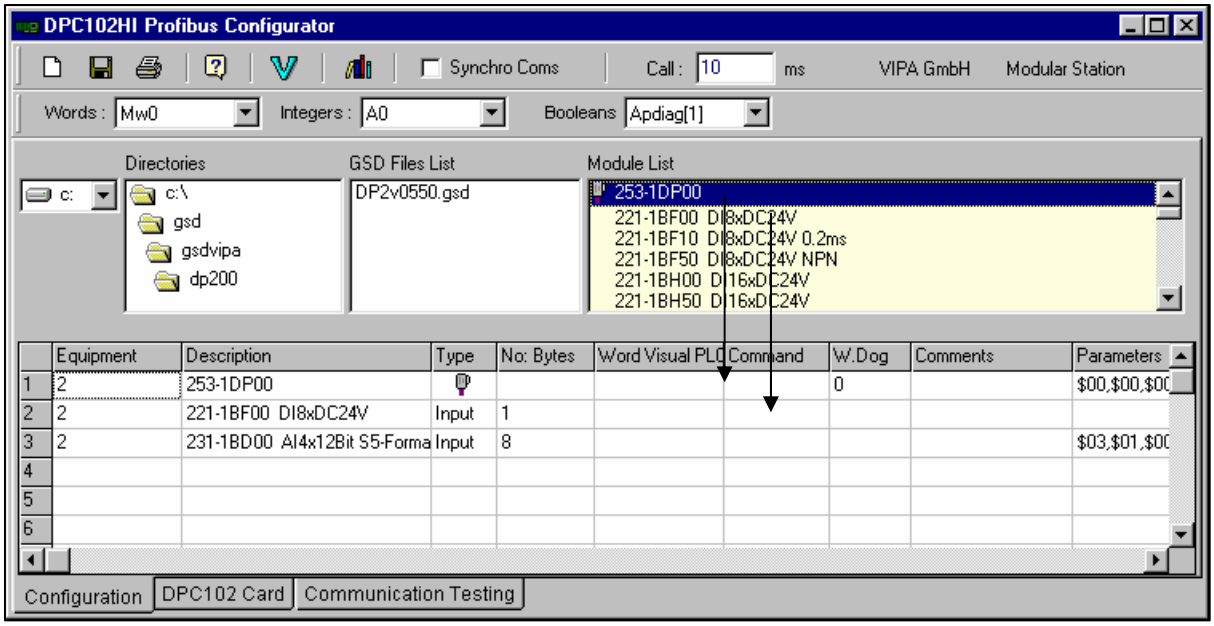
Send the command by hitting the 'Enter' key.

PROFIBUS Configuration – Card DPC102



The configurator is an external program like other configurators, (Confmpi and AS511) except Modbus which is integrated.

The PROFIBUS setting used some files called commonly GSD files. Choose a GSD file (extension .GSD) by navigation on your disk. The list of the modules is displayed in the right hand listbox (yellow background)
Bring the interface module by double-click or drag & drop, and then the type of modul you need.



According to the chosen module type, the number of bytes in input and output is displayed. You can inform the user of the choice of the visual I/O variable (Or Visual PLC) to register in the column Word Visual PLC.

Word Visual PLC (or Visual I/O): First word receiving bytes read or written from the profibus equipment.

Command : If no bit is specified, the interrogation of the equipment is made regularly by the card. If a Visual I/O bit is specified, the interrogation of the concerned equipment is provided when this bit (Boolean) is set. When the frame has been read (the equipment replied without error), this bit is set to 0 (False) by the card which returns a reception acknowledgement.

W.Dog : WatchDog. If no value is specified, Profibus outputs remains in the state when any frame is sent by the card. If a value is specified (x100ms), a frame has to be sent regularly by the card before reaching this value.
If it is not the case, the outputs are reset and the station passes in Bus Fail State.

Comments : Your comments.

Parameters : Enables to display or to modify the initialisation parameters of the station (doubleclick on cell).

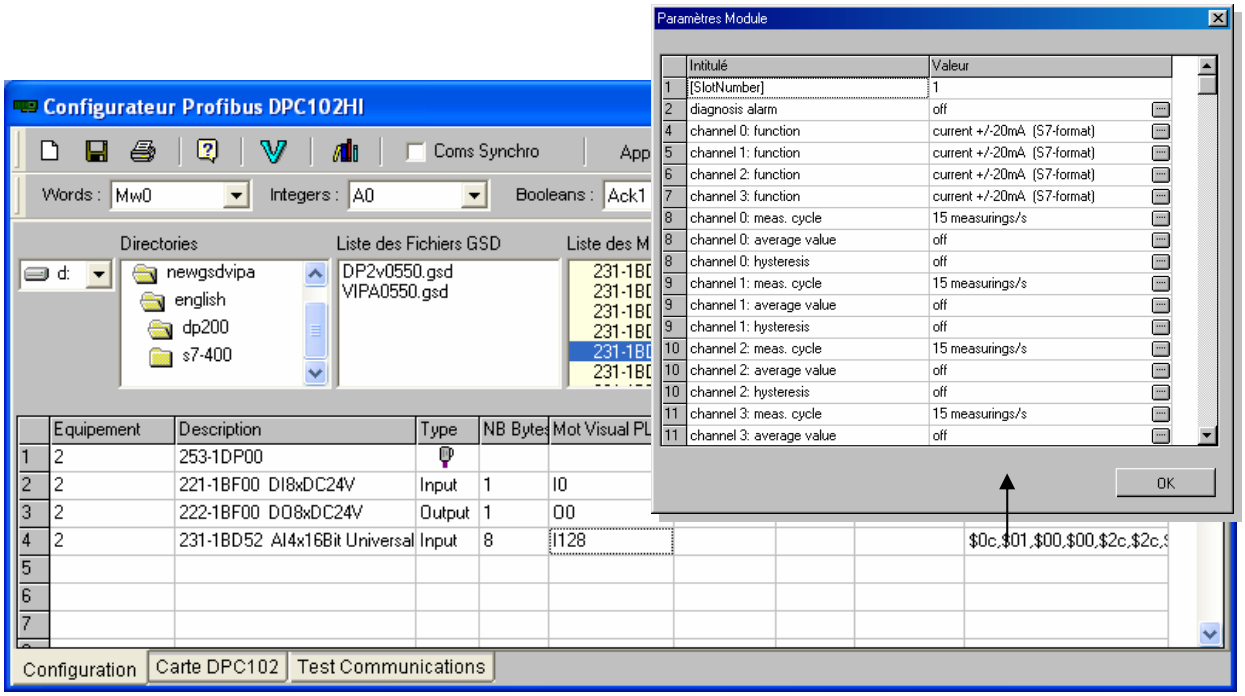
Coms Synchro : If this checkbox is checked, the 'PCARSDP.VPU' driver is different, allowing to **chain** all profibus equipment every 10 ms systematically. If Coms Synchro is not checked, each equipment is scanned one by one every 10 ms (or 2ms mini).

Save, the configuration tool compiles a driver according to your configuration. You must put the VPU **PCARSDP.VPU** in the real time engine of Visual PLC if you want to start the communication.

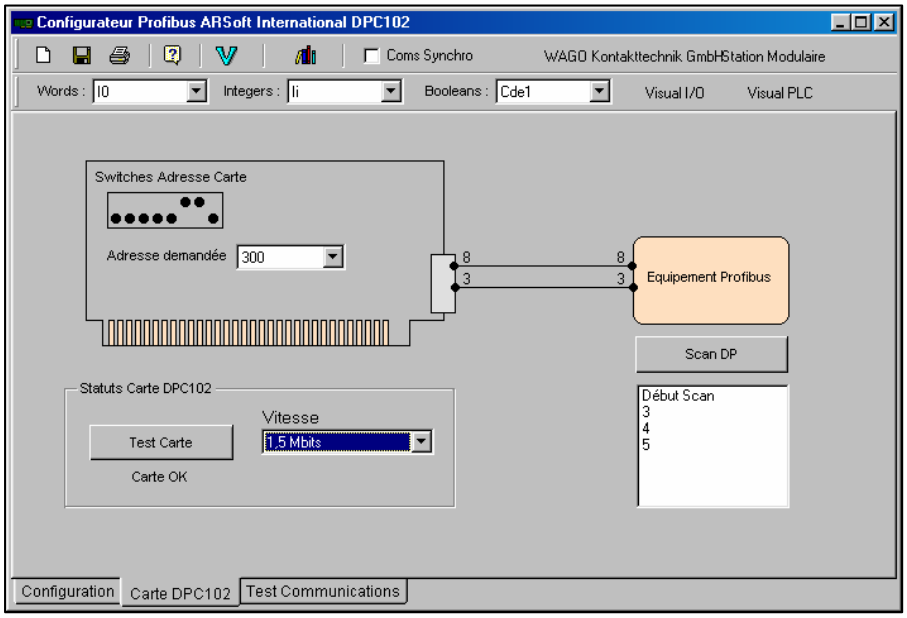
For Visual I/O put **PCARSDP** in Constitution Application/Tasks priority.
.

Setting the equipment parameters

Double-click on the **parameter column** to set each parameter of the concerned module. Below a module of 4 analog inputs where you must set each canal (4/20ma, 0/10V, PT100 etc..)



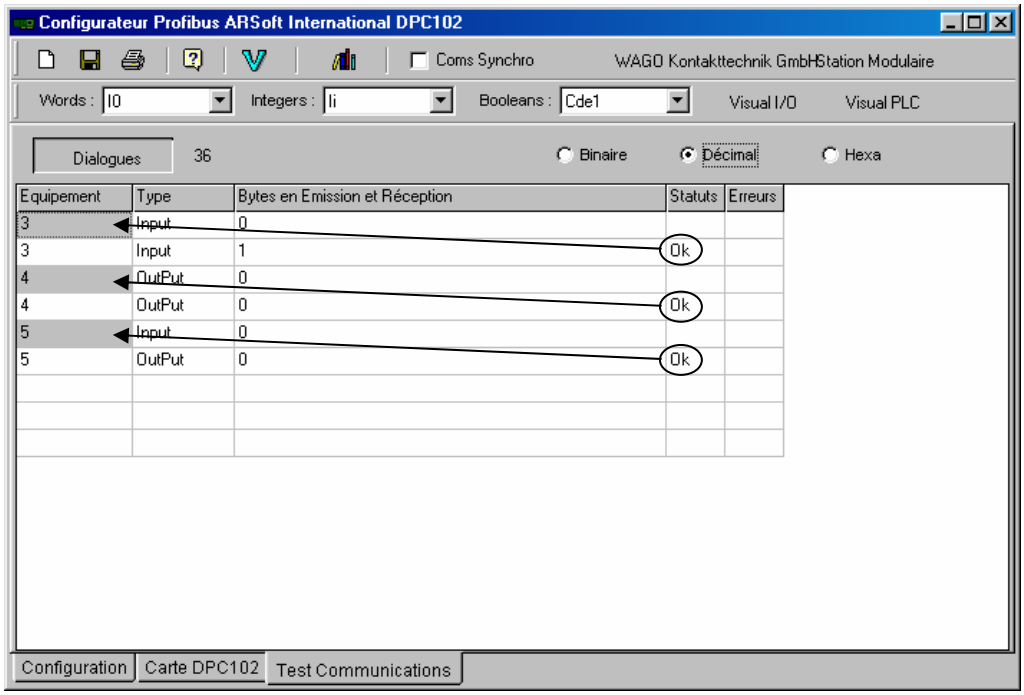
Test of the Profibus card and ScanDP.



Click on the button Test Card to test the profibus card ('loop Back' Mode) completely.
This command shows the position of Switches on the card. If the card is not detected, it is probable that the card is in address conflict with other card on the bus or the driver is not installed. Choose an address in the Combo to search for an adress', the system indicates you the new position of Switches.

Scan DP: Allows to scan connected equipments on the Profibus bus. Because of the speed self-adaptation, slaves do not reply immediately.
This command must be done several times if necessary.
Some slaves memorize the last speed. It is necessary to shutdown power supply to change it

Testing the communication with connected Profibus equipment



Click on the button Dialogs, a grid appears displaying the value of the bytes read and written in the equipments. You can force some bytes by writing a new value in the concerned cell.
Cells with a grey background in the Equipment column indicate the first byte of the equipment.

It is evident that you can only force the bytes with 'OutPut' statement.
If the equipment replies normally, the text Ok appears, otherwise the number of detected errors appears in the error column.

Here the first text OK starting to the top corresponds to the equipment #3

Dangerous Interactions with Visual PLC

The PROFIBUS configurator is an external program that generates and compiles a VPU that is the Profibus driver. This driver **PCARSDP.VPU** is usable by the real time engine of Visual PLC and by Visual I/O. In the case you use Visual PLC and Visual I/O together, it is necessary to have only one PCARDSP.VPU active on your PC.

It is necessary that PCARSDP.VPU is in the priority tasks of Visual I/O or in the real time engine of Visual PLC BUT NOT IN BOTH.
Generally, the Profibus driver concerns the automation in Visual PLC.

Necessary files to use the Profibus card.

DPConf.DP : The configuration file (contents of the grid) in the application directory.
ARSBT.SYS: The Windows driver. It is in the directory Windows/drivers32
Arsdp.dll : The DLL interface with the configuration file in the directory Windows.

☞ **Note**: In the case of Arsoft panel PC file, DPCONF.DP is not necessary in run any more. The parameters are included in the Profibus driver.

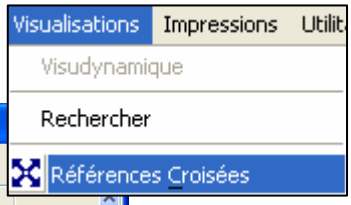
Coms synchro: You have the possibility of changing the frequency of driver's activation and passing to 1 ms and multiples. However, considering that Profibus equipment reacts into 1 ms, it is impossible to scan more than one Profibus slave within 1 ms.

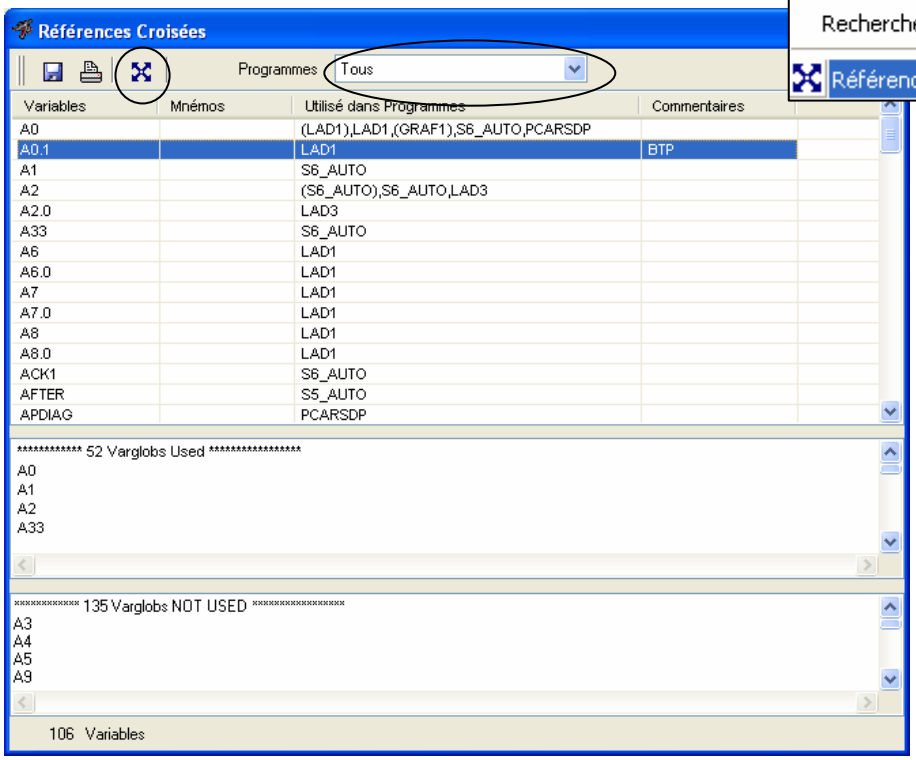
It is necessary if you want to read Profibus slaves quickly which are set the driver to 1 or 2 ms and do not set Coms synchro.

Cross references

The cross references is regenerated at compilation time of the forms and the programs.
The compiling creates file with extension .REF which are the result of founded variables by the compiler.

Click on cross references, a grid is displayed:
Choose a program or "'All", and then click on the button below.





The higher grid fills the variable column that can contain a word (A0) or a bit in a word (A0.0) in the case of entireties (Byte, Word and integer).

Mnemos : Are not used in VisualI/O. Only Visual PLC can define the mnemonic for the variables.

Comments : Are not used in VisualI/O. Only Visual PLC can define comments for all the variables.

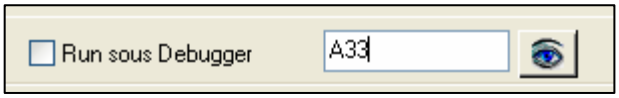
Used in programs
Example A0 : (LAD1),LAD1,(GRAF1)

Means that A0 is affected (directly in word or a bit in the word). It is the program named LAD1and GRAF1 that write this variable because the program name is in between brackets.

Two listboxes **display** the **global** variables used in the programs and those not used**.**

Localisation of a variable in a form.

This functionality is operational only on figures coming from the first 2 tabs in the toolbox.



Strike the variable sought, and then clicks the button on the right. The figure containing the variable requests blinks on the form.

Visual I/O Components

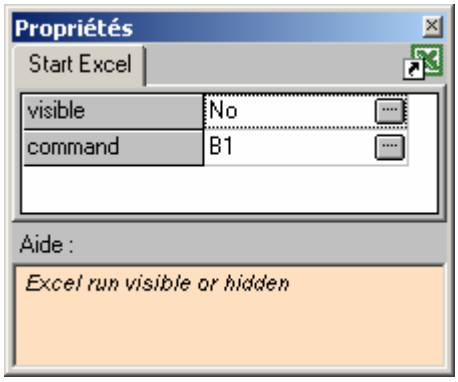
Excel Tab

- Open and close Excel
- Load an excel sheet.
- Create an Excel sheet.
- Read and write in cells.

The Excel components are 10. They allow the installation of an OLE link between Visual I/O and Excel. They become invisible at launching time of your application. The last component is enabled to constitute an Excel file using the global variables values.

StartXLS Component

This component is enabled to launch Excel visible or hidden from your application.

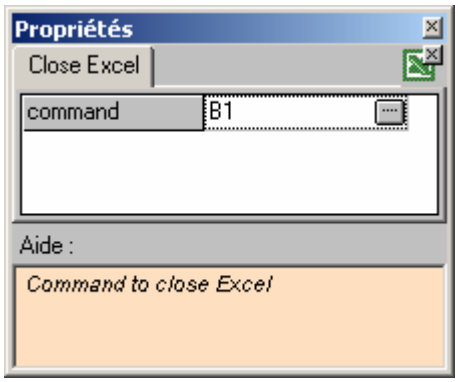


Visible : 'Yes' or 'No' Launch Excel with visible or hidden attribute.

Command : Variable to start Excel application.

StopXLS Component

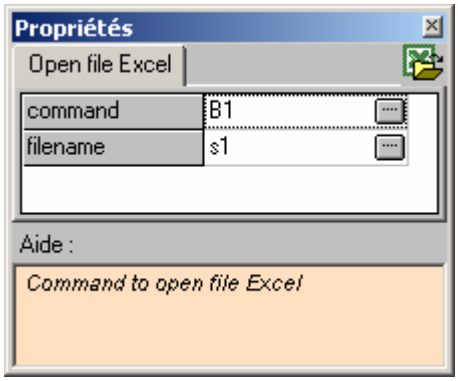
This component allows closing Excel from your application.



Command : Variable to close Excel.

OpenXLS Component

This component allows opening an excel file from your application.

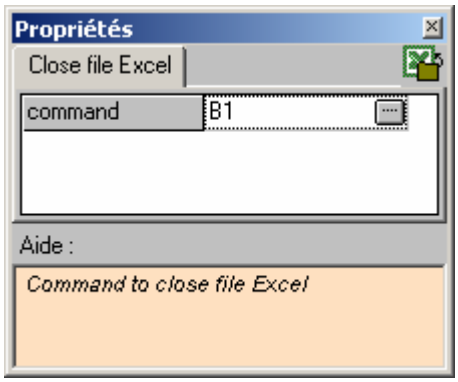


Command : Variable to open the Excel file.

FileName : XLS file name to be opened. If a variable with an empty String (` ') is passed in parameter to the component, a dialogbox is displayed to choose the file. Then the variable takes the name of the file selected in the dialogbox.

CloseXLS Component

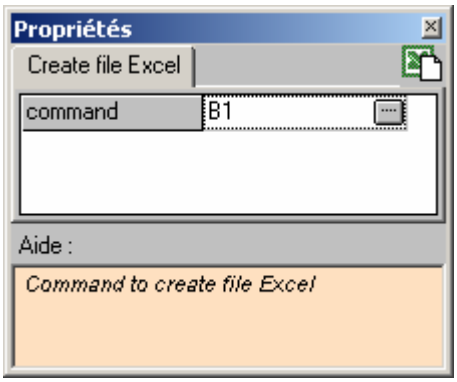
This component allows closing an Excel file from your application.



Command : Variable to close the file.

NewXLS Component

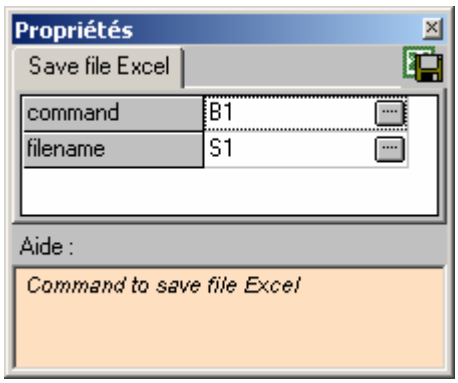
This component allows creating a new grid in Excel from your application.



Command : Variable enables creating an empty Excel sheet.

SaveXLS Component

This component allows recording the Excel file opened from your application.

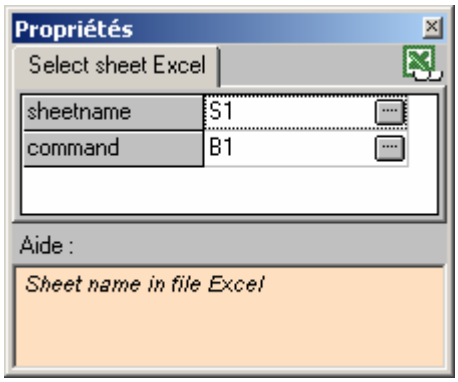


Command : Variable to start recording of the opened file.

FileName : XLS file name to save. If a variable with an empty String (` ') is passed in parameter to the component, a dialogbox is displayed to choose the file. Then the variable takes the name of the file selected in the dialogbox.

SheetXLS Component

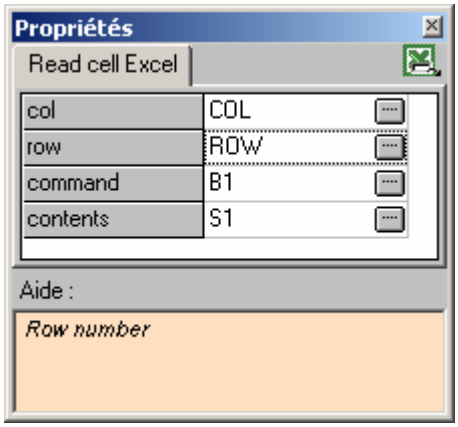
This component allows selecting an opened sheet from your application.



SheetName : Sheet name to be opened in the file name selected.
Command : Variable to start the selection of the sheet.

ReadXLS Component

This component allows reading the contents of an Excel cell from your application.



Col : column number of the cell to read

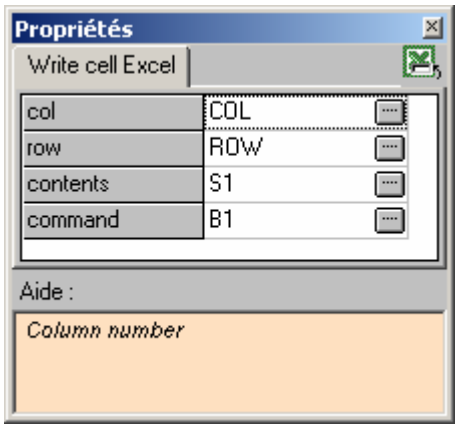
Row : Row number of the cell to read

Command : Variable to start the reading of the cell.

Contents : Result of the Cell reading. This variable is a String type.

WriteXLS Component

This component allows writing in an Excel cell from your application.



Col : column number of the cell to write

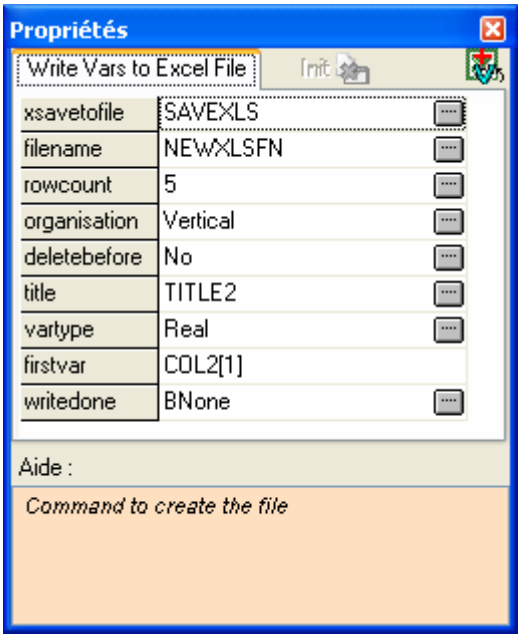
Row : Row number of the cell to write

Command : Variable to start the writing of the cell.

Contents : The content of this variable is written in the excel cell. This variable is a String type.

ColVarToXLS Component

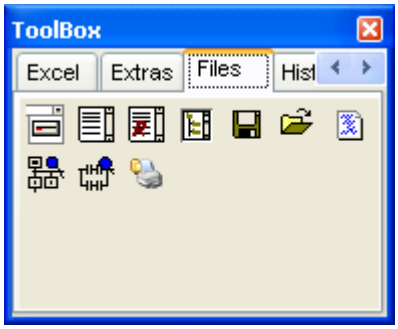
This component allows filling a column or a line in a compatible Excel file with the global variables consecutive values (It is simply necessary to specify the first variable, its type and the number of consecutive variables to this one). To constitute several columns in a XLS or CSV file, you must use several components ColVarToXLS. Each component treats a complete column with global variables. If the Excel file must contain 5 columns, it is necessary to use 5 distinct components.



XSaveToFile : Command to save the file with the name specified in **FileName**.
Rowcount : Number of cells written with consecutives variables into vertical or horizontal direction.
Organisation : Writing in a column (Vertical) or in a row (Horizontal).
DeleteBefore : Yes allows to erase the final file before the treatment. Set Yes for the first column or row.
Title : Column or row Title. If it is an empty string, no title is added to this one.
Vartype : Type of the series of variables to be displayed in the column or row.
FirstVar : Name of the first variable written in the column or the row. The other variables are those declared consecutively in the global variables.
WriteDone : Optional bit set when the writing is performed.

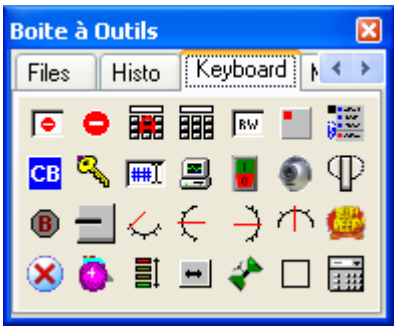
Files tab

- Displays all the drives available.
- Lists all the files with a specified extension.
- Directories list
- Display a text file.
- Display a "Save As" dialog for saving files.
- Display a file-selection dialog.
- Display a Grafcet
- Display a Ladder diagram.
- Printer Settings



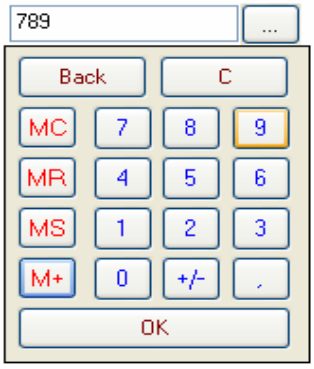
Keyboard Tab

- Edit Password.
- Edit Password.
- Alpha Keyboard
- Numeric Keyboard
- Read/Writeeditbox.
- Design Button.
- Neutralization desktop.
- 2 colors button.
- Access level management.
- Mask editbox.
- Neutralization Ctrl/alt/del
- Switch.
- Webcam Interface.
- Switch 3 positions.
- Small zone bouton type.
- Vertical Slider.
- 4 Gauges
- PC Buzzer On/OFF (on motherboard)
- Bitmap Button for closing the form.
- Animation with successive bitmaps.
- Bargraph with 3 color level.
- Zone to move the form.
- Display several Bitmaps according several bits.
- Mouse clic zone.
- Editbox with numeric keyboard.



Editbox with numeric Keyboard

This component allows showing and editing a numerical value (Integer or real). By clicking the button to the right of the editbox, a numeric keyboard is displayed allowing hitting the numerical value in the help, for example, with a touch-sensitive screen. MS and MR keys are Memory Clear and MR Memory Recall which erase the internal memory and call back the value from the memory. M+ key is enabled to add the current value to the internal memory of the component.
The MS key allows storing the current value in memory.

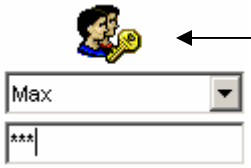


If you want to modify the component, it uses the form NUMCLAV.SYN located in the subdirectory modules (C:\API32\Modules). Copy all files which begin by NUMCLAV (NUMCLAV*. *) to your working directory.

Modify the keyboard as a normal form, and then send back all the files (numclav*. *) into directory Modules.
Then recompile the component (click right component / creation)).
To validate the value entered immediately, always select the attribute
Else to validate the value only by clicking the Ok button, select the On return only attribute in the properties box.

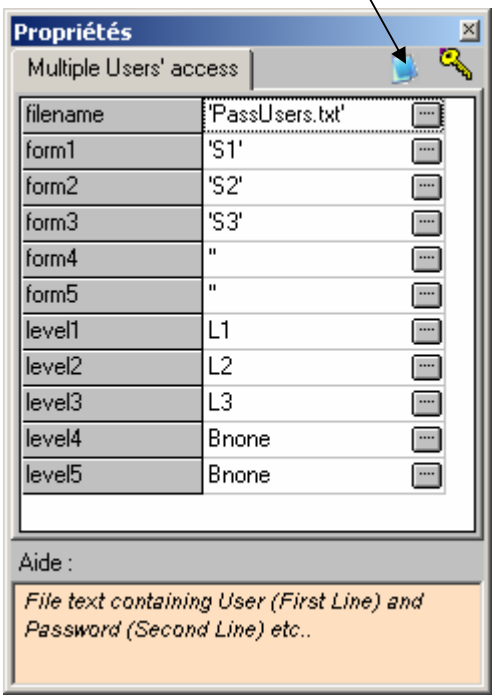
UserAccess Component

Component UserAccess allows the installation of various levels of access in your application. The levels of access and the corresponding passwords are stored in a text file. Each level is associated to a Boolean variable *Level* allowing showing or hiding certain parts of your synoptic or prohibiting access under certain menus. Each level is associated to a Form variable of the String type, which contains the name of form called.



A double click shows a ComboBox to choose the type of user and EditBox to edit the corresponding password.

Click here to display the notepad to declare the names of the users or the levels and the associated passwords.



The text file containing the users and the passwords are defined in the following way:
The first line defines the user selected at the launch time of the application.
The second line defines the password of the first user.
The third line defines the second user.
The fourth line defines the password of the third user. Etc...
It is possible to declare users up to 5.

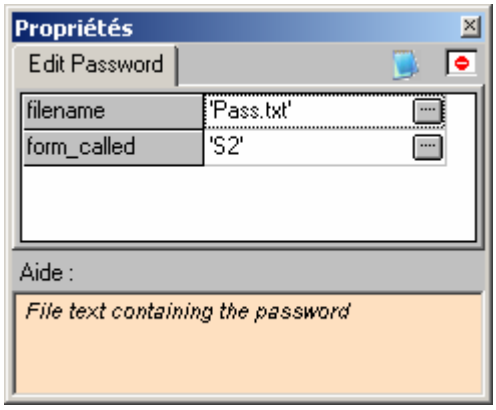
Example of text file user:
Maintenance
123
Operator
456

FileName : Name of the file containing the various users and passwords.
FormX : Name of synoptic called after editing the password corresponding to user X defining in the file.

LevelX : Boolean Variable (set) after editing the password corresponding to user X defining in the file.

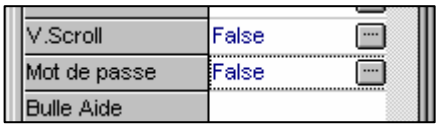
EditPassword Component

EditPassword Component allows the control of a password to Display a synoptic.
The password is stored in the first line of a text file.



FileName : Name of the file containing the password..

Form_Called : Name of the form displayed if the Password edited is correct.



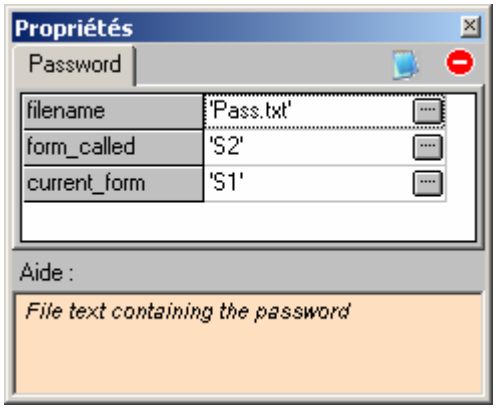
It is possible to disable passwords hidden under asterisks in the properties of the EditBox (F6).

Password Component

Password Component allows the installation of a password for displaying a form. The password is stored in the first line of a text file. This component is composed of an editbox for the password, a button for validation and a button for exit.

This component is used in a form, which will appear when an access request is needed in your application.

Exaemple : `If Button Then Display('PassWord') ;`



FileName : Name of the file containing the password.

Form_Called : Synoptic called after validation of the password by the button `Ok'.

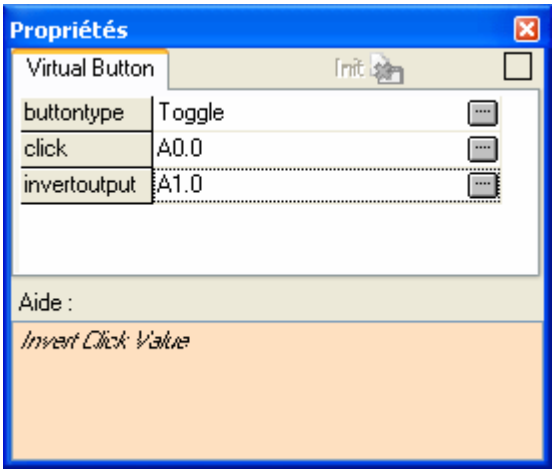
Current_Form : Synoptic containing the component Password. This synoptic is automatically closed after validation of the password by the button `Ok' or cancelled by the button ` Cancel'.

Move form component

This component allows moving the form by clicking in the zone, and then moving it with the mouse.
The action to be made is similar to what is made in the title bar of the form.

Virtual button component

This component shows a rectangle allowing the operator to click above to use it as of a classic button.
This component allows you to transform whatever zone of the screen into button.



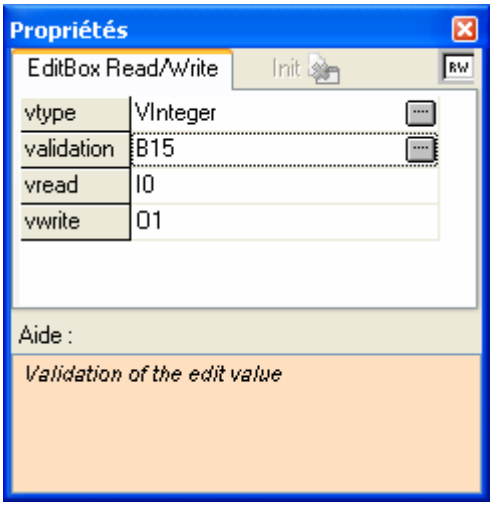
ButtonType : Simple or toggle. (Simple or toggle).

Click : Boolean to affect in case of click left with the mouse or action on the touchscreen.

InvertOutput : Boolean whose state is the opposite of click.

EditRW Component

Component EditRW allows using EditBox in reading and writing. This EditBox may be used with the various types of variables in Visual I/O. Reading and writing can be associated to two different variables.



VType : Allows defining the type of variables used by the EditBox. (VWord, VReal, VString, etc...).

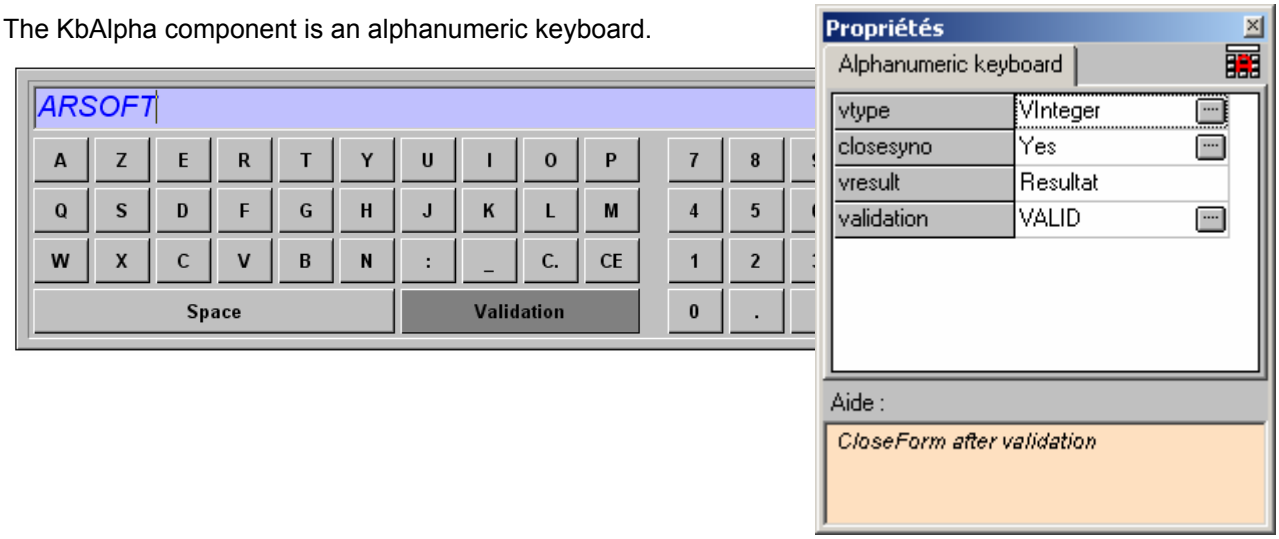
VRead : Variable displayed in the EditBox (Reading).
VWrite : Variable set by EditBox (Writting).

Validation : Variable of validation of the editing in the EditBox. If the variable passed in parameter is BNone, the validation is done by pressing the Enter key.

The validation of the EditBox contents is made when the focus leaves the editbox or when the operator strikes the < Entered > key or the bit validation passes to True.
☞ **Note**: The display of the value contained in the VRead parameter is done only when this variable changes value or when the form is displayed (on winapparition).

KbAlpha Component

The KbAlpha component is an alphanumeric keyboard.



VType : Allows defining the type of variable used by the keyboard. (VWord, VReal, VString, etc...).

CloseSyno : After validation, allow closing the form which contains the alphanumeric keyboard.

VResult : Variable which will receive the result of the editing of the alphanumeric keyboard.

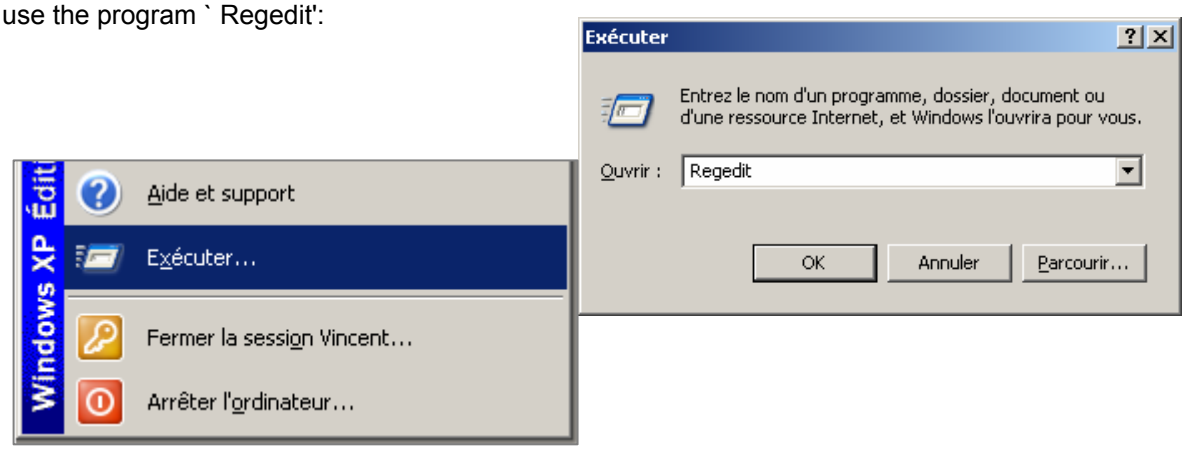
Validation : Hitting the key ` Validation' of the keyboard to set the variable `Validation'.
The Reset of this one must be reset by program or automatically by component EditRW which has focus.

KillExplo Component

KillExplo hides the Windows' explorer (desktop) at the launching time of your application.
The Reload parameter allows starting again the Windows' explorer when you close your application.

Reload: This parameter allows restarting the Windows Explorer or not when closing your application.
If the Reload parameter is Set (True) or if the BNone variable is associated to this parameter, the Windows' explorer is started again when closing the application..

Attention : The Windows operating system is parameterized to restart the explorer automatically when it is closed. It is necessary to modify the key in the Windows registry.
use the program ` Regedit':




```
HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\WINDOWS NT\CURRENTVERSION\
WINLOGON\AutoRestartShell = 0
```

KillExplorer Component launches **KillExplo.exe**, which must be present in the directory of your application or the C:\Windows Directory or the C:\Windows\System32 Directory.

Ctrl_Alt_Del Component

Ctrl_Alt_Del component allows prohibiting the combination of keys giving access to the task manager of Windows during your application. This component does not require any parameter.
It becomes invisible at the launching time of the application.

Attention
This component must be placed on the first form displayed at the launching time of your application to be executed on the first cycle program.

Must be set in the registry (launch Regedit.exe):

```
HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\
WINDOWS NT\CURRENTVERSION\WINLOGON\AutoRestartShell = 0
```

Mini recipes tab

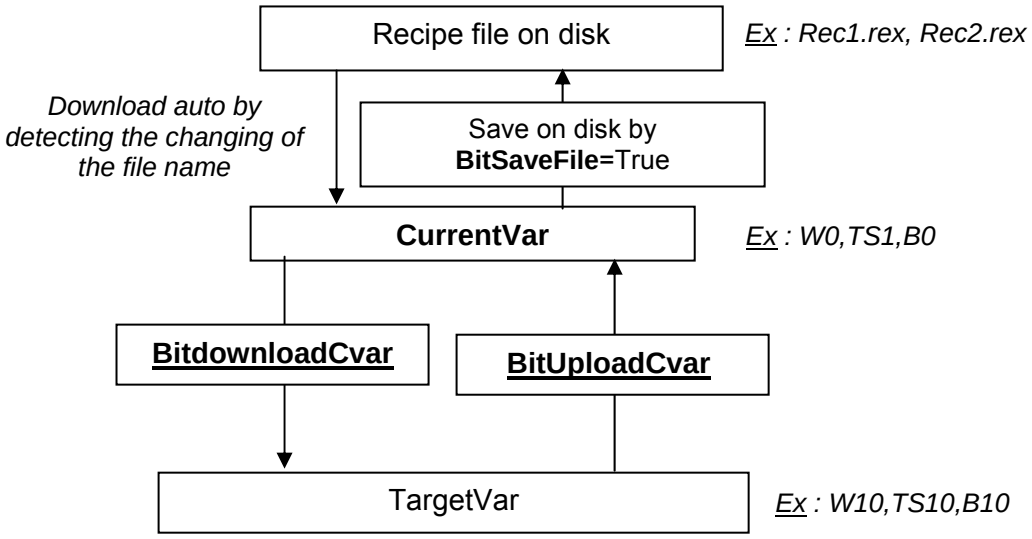
- Numerical and alphanumeric editbox.
- Radio button
- Checkbox
- Images
- Management and files navigation.

These components allow saving numerical alphanumeric or Boolean values (radioboutons, check box) and images' name in a file. It is possible to create a recipe by means of file name affecting the variables of the process. For example, a selection of a recipe can be created by means of a selected file name in a list.

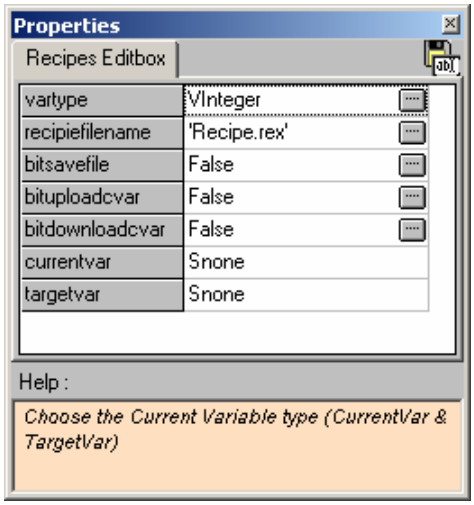
Principle

We work on a variable, which can be displayed and modified. This variable can be the variable in the process itself or an intermediate variable, which will be assigned to the process by a mechanism (**Download**) or will be got into the process by a mechanism (**Upload**).
In the case of a dowload of a value from a file, the value takes the following way: File to ther currentvariable then if the bit downloaded is true, this value is also assigned to the final variable process.

In the case of an uploading value to a file: The value is read in the process (variable process), and then it is transmitted in a current variable which itself is safeguarded in a file.



Editbox for recipe



Editing and numeric visualization of variable or string.

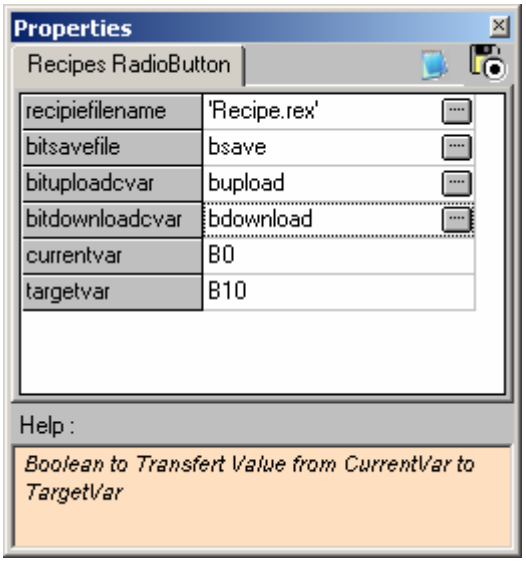
Vartype : Type of variable in Visual I/O.
RecipeFileName : Name or variable containing the name of the backup file.

If this name changes, the editbox is updated automatically with the new vlaue.

BitSave : Boolean to save the value (currentVar) in the file.
BitUploadCvar : Allows updating the value of targetvar to the variable *currentvar*
BitdownloadCvar: Allows sending the value of the CurrentVar variable into targetvar.

CurrentVar : Working Variable of *Vartype* type.
Targetvar : Allows receiving or sending values in currentvar.

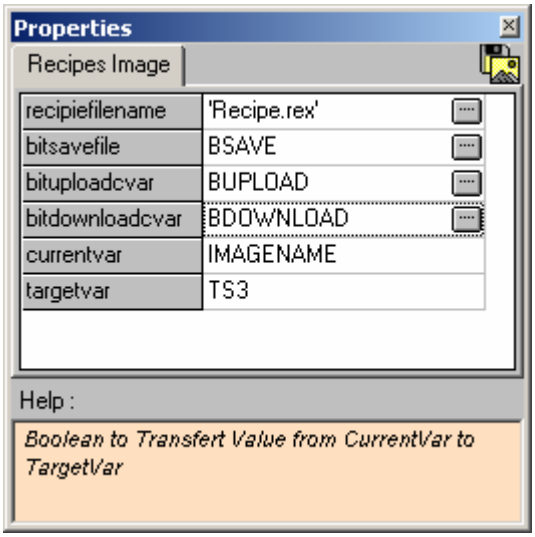
Radiobutton and checkbox for recipe



Component running in the same manner of the others

Currentvar and targetvar must be Boolean type.

Image for recipe



CurrentVar contains the name of the image to be shown. This name must contain the complete image path.

Ex C:\mybmp\Photo1.bmp

☞ **Note** : These components are enabled to produce simple forms. These recipies run only if the form containing these components is displayed. It is impossible to download or safeguard the values by program. Forms like database can be made up

The other components

FileList Returns the list of the files of a specified extension without possibility to refresh this list.

FileList with refresh Returns the list of the files of a specified extension with possibility to refresh this list by a bit.

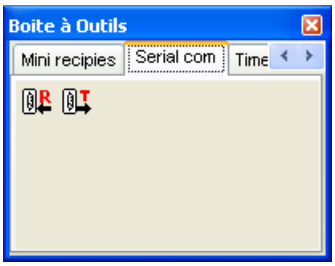
Directory listbox : represents a list box control that is aware of the directory structure of the current drive

Drive combo box : implements a specialized combo box that displays all the drives available when the application runs.

An illustration of these components is made in the rubric*: To display a textual file in an editbox*

Serial link tab

- Reception of Characters.
- Emission of Characters.



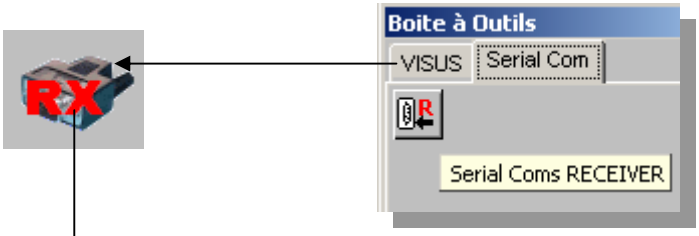
2 components simply allow receiving or emitting characters on a serial link. It is more usable than the functions of the Serial.vpu library.

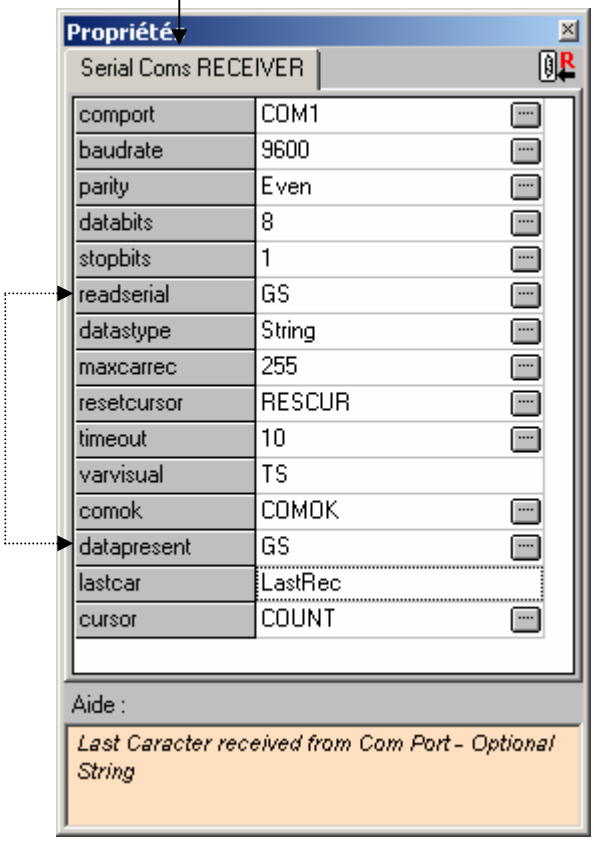
Serial Coms Receiver

Component description

- Initialization of the serial communication port COM1 to COM8.
- Characters reception and storage in a Visual I/O variable.
- Management of the communication breaks down (Time Out).

Attention : If that component must be operational, the container (form) must be visible. If this component must always be in service, place it in the principal form of your application (first form displayed at launch time).





Serial comport settings

Comport : Com1 → Com8
Baudrate : 110 → 115200 bauds
Parity : Even, Odd, None
DataBits : 8 Bits or Bits of data
Stop Bits : 1 or 2 stop bits

Functionalities

ReadSerial : Input enables reading and storing the recept characters on the comport. Connecting this input on the output **Datapresent** then a read is performed at each character reception.

DataType : String or Byte. Describe whether the characters are ASCII or numerical.

ResetCursor : Enables to reset the reception pointer..

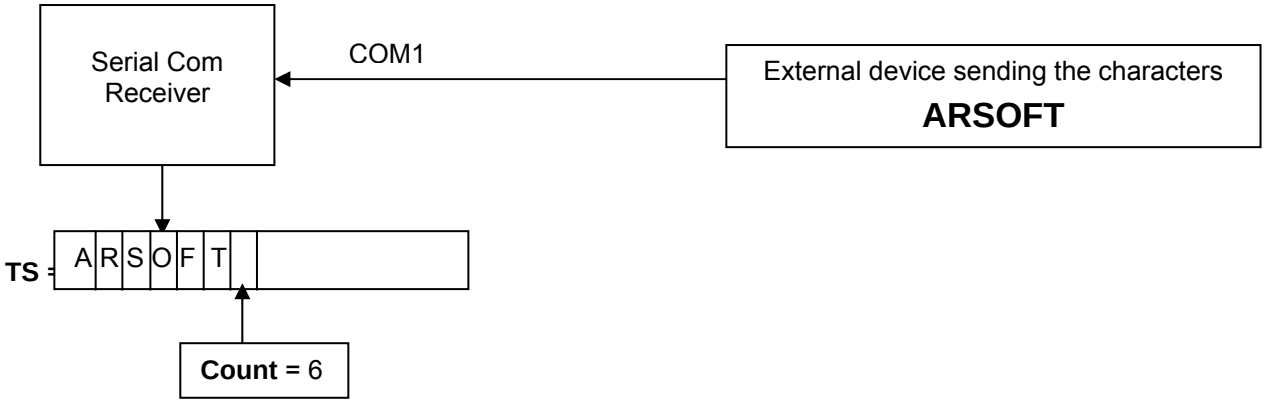
TimeOut : in x10ms enables the setup of a maximum time between 2 characters reception. If this time is overlap, a reset of both the cursor and the Visual IO variable is done. If time out=0, no treatment is perform.

VarVisual : Visual I/O Variable in String type or Array of Byte.
Receiving all the characters coming from the comport.
ComOk : Boolean set (True) if the comport is opened at the launching time of the application.
DataPresent : Boolean set (True) if a character non read is present in the comport buffer (Windows buffer).
LastCar : Byte of the last character received.
Cursor: internal Cursor counting the character in the Visual I/O variable.

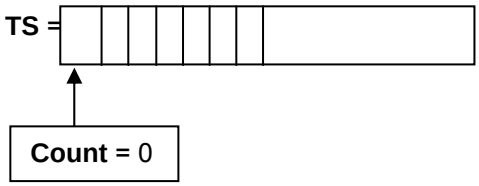
Example Global Variables

GS : **Boolean**;
ResCur : **Boolean**;
TS : **String**;
ComOK : **Boolean**;
LastRec : **Word**;
Count : **Integer**;

Buffer of Visual I/O



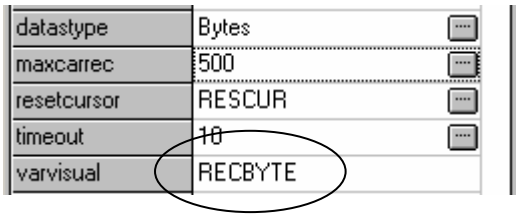
If ResetCursor = True or Timeout>0 or Cursor>=255



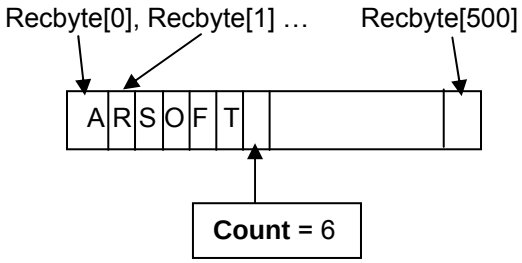
The principle remains same with an array of bytes

Declare an array in global Variables

RecByte : Array [0..500] Of Byte;



Here **DatasType** is set in reception of bytes and the Visual I/O variable is an array receiving the characters of the serial link.
The maximum is set the array size (here in our example 500 bytes).



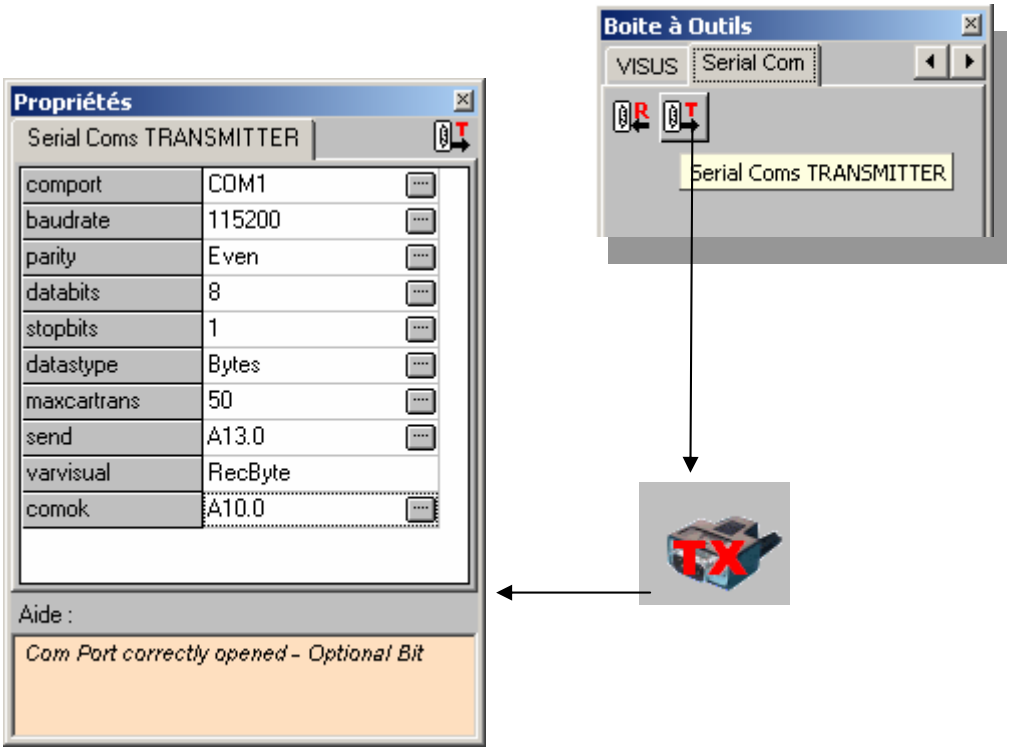
Serial Coms Transmitter

File name Com232TX.VPU

Component description

- Initialization of the serial comport COM1 to COM8.
- Characters emission on a serial comport.

Attention : If that component must be operational, the container (form) must be visible. If this component must always be in service, place it in the principal form of your application (first form displayed at launch time).



Serial comport settings

Comport : Com1 → Com8
Baudrate : 110 → 115200 bauds
Parity : Even, Odd, None
DataBits : 8 Bits or Bits of data
Stop Bits : 1 or 2 stop bits

Functionalities:
DataType : String or Byte. Describe whether the characters send are ASCII or numerical.

MaxCarTrans : Number of characters to send on the comport. This parameter is useful only in the case of bytes emission. In a case of String variables emission, the length of the string is automatically used.

Send : is enabled to send characters on the serial comport. The sending is done on rising edge, the user program must reset the bit and set it again to perform a new sending.

VarVisual : Visual I/O Variable (String or Array of Byte type) sent on serial link.

ComOk : This Boolean is set if the comport is opened correctly at your application launching time.

Instructions for the management of the serial link – Serial.VPU Library

You can also directly control the connection by using the functions and procedures of the Serial.VPU library.

```
// Open a serial connection
Function OpenCom (ComPort,Baudrate,Parity,Bits,Stop:String ): Boolean;

// Test whether the serial channel specified is already open
Function  GetComOpen(Comport : Integer) : Boolean;

// Close a serial comport
Procedure DisableCom (ComPort  : Byte);

// Close all serial comports
Procedure DisableAllComs;

// Test whether a character is present in the reception buffer
Function V24DataOk (ComPort  : Byte) : Boolean;

// Returns one character from the reception buffer
Function GetByte (ComPort  : Byte ) : Byte;

//  Send a String in the emission buffer
Procedure ComWrite (ComPort  :Byte; Value : String);
```

Example Portion of code

```
Uses Serial, Sysplc ;
Var
  Ok        : Boolean ;
  DoClose   : Boolean ;
  B         : Byte ;
  TS        : String ;

Begin
  Ok := OpenCom('1','9600','E','8','1') ; // Try to open the comport
                                          // Parity = Even, 8 dataBits & 1 stop Bit
  If GetComOpen(1)=False Then Message('Error Com1 not open');
  If DoClose And Ok Then DisableCom(1) ; // *** Close the serial comport
End ;

Begin
// ** Test whether there are characters in the buffer ****
 If V24DataOK (1) Then // *** Test presence chars in com1
 Begin
    B  := GetByte(1) ; // *** Get one char from the buffer
    TS :=TS+Chr(B) ;   // *** IF it is an ASCII char, add it to the string TS
    SendByte(2,B)      // *** Send the byte on Com2
    ComWrite (2,TS) ;  // *** Resend the entire string on com2
 End ;
End ;

// Send bytes to the emission buffer
Procedure ComWriteBuf (ComPort: Byte; Var PointerTable; NB: Integer);
```

Example

```
 Var
    Tab: Array [0..100] Of Byte ;
 Begin
   ComWriteBuf (3,Tab[0],101); // ** Send 101 bytes on COM3
```

Times Tab

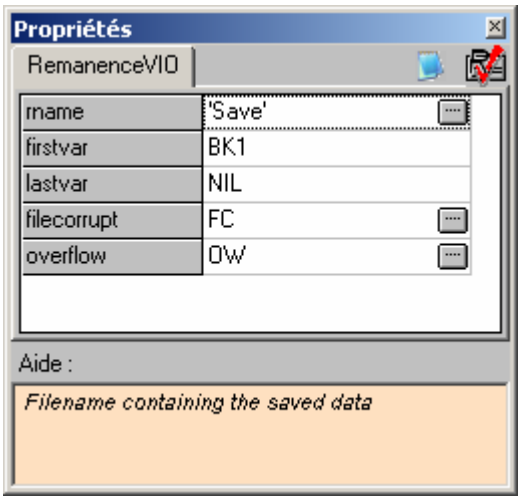
- Data Remaining.
- Time base.
- Display Date and Time
- Calendar.
- Chronometer
- Detect changing values
- Switch to a specific language.
- Temporization

RemanenceVIO Component

The component RemanenceVIO allows saving the useful data of the process and restoring them at the lauching of the application. The data are compressed and stored in a file.

Example :
Let us suppose that variables BK1, BK2, BK3, BK4 are declared consecutively in the global variables.

```
BK1   : Integer ;  // backup Variable
BK2   : Real ;     // backup Variable
BK3   : Boolean;   // backup Variable
BK4   : String;    // backup Variable
RIEN  : Boolean;   // Heel
```



RName : Name of the backup file.

FirstVar : First variable of the storage block.

LastVar : Last variable + 1 of the storage block.

FileCorrupt : Indicate an error in the file during the restoration of the data. This variable must be reset by the user program.

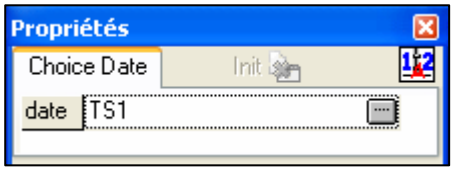
OverFlow : Indicates an overflow. The maximum size of the data compressed is reach. The variable must be Reset by the user program

The component Remanence save BK1 to BK4 in a file named 'Save'. The Rien variable is not safeguarded.
The safeguard is made from variable ` FirstVar' to variable ` LastVar' -1.

Attention :
The component Remanence must be placed on the first form displayed at the launching time of your application (see the functions Display and DisplayModal).
If you intercalate variables in a block which has already been safeguarded, these additional variables will take the values of the old variables present at this place.
It is imperative to insert the new variables right before the `Heel'.

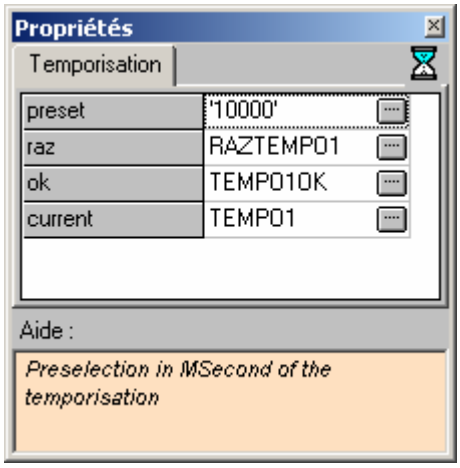
The components **Display time** and **Display Date** display respectively the time and the current date.
These 2 components don't have parameters.

The component **ChoiceDate** enables selecting a date thanks to the Windows calendar.
This component needs a string variable receiving the selected date.



Temporisation Component

This component allows the installation of temporizations in your application. The unit of used time is the millisecond.



Preset : Preselection of the value of temporization in millisecond.

Raz : Reset of the temporization.

Ok : State of the temporization (True if terminated).
Current : Temporization current value. (Integer Type).

Attention :
Component Temporization is active only if the synoptic that contains it is displayed on the screen.

A TempoMS function identical to component Temporization is available in the Sysplc library. It may be used in form or in a cyclic program.
.

Procedure Temporisation *(Preset: Integer; Raz: Boolean;Var ok: Boolean; Var Current: Integer);*

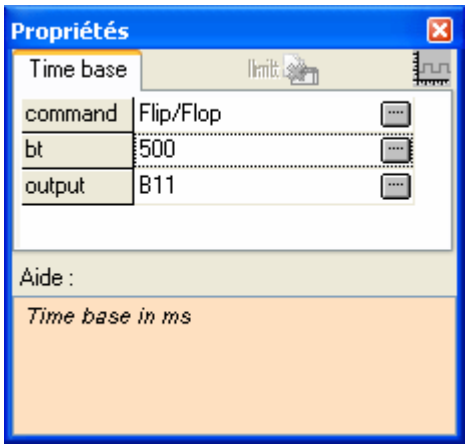
Example of the use of the procedure Temporisation in a form :

```
Uses Temporisation;
Var
   Tempo10 : Temporisation;
//*** Circle1 ***************************************
SObject Procedure Circle1;
Begin
   Tempo10(5000,RazTempo1,Tempo1Ok,Tempo1); //** Preset,Reset,Ok,Current
   If Tempo1Ok Then SetFillColor($00FF00)
               Else FirstColors;
End;
```

TimeBase Component

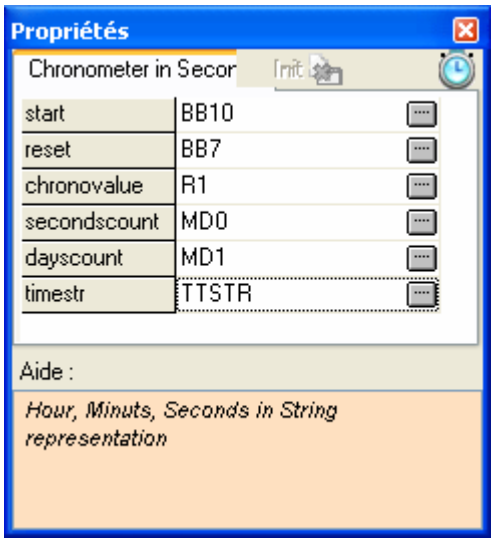
To create a timebase that allows invert a bit (Flip/Flop) to set it (True), to reset (False) or to perform a pulse on a cycle tour (Pulse).

Command Flip/Flop, Set, Reset or Pulse.
Bt Constant or variable in ms.
Output Bit to affect at the end of the temporization requested.



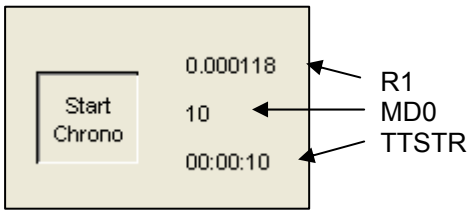
Chrono Component

To start a chronometer.



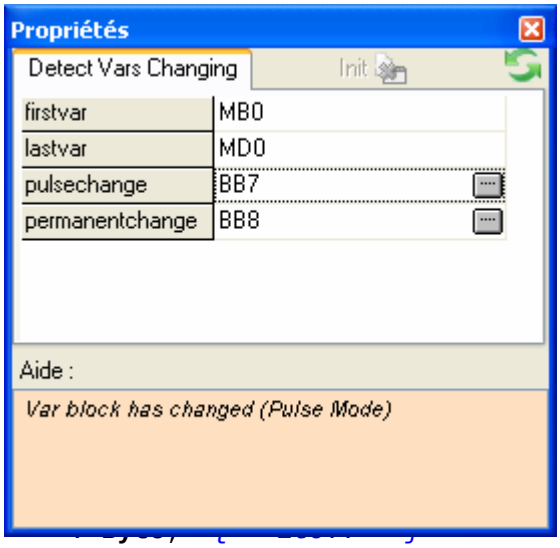
Start : Boolean to start or stop the chronometer.
Reset : Boolean to reset the chronometer.
Chronovalue : Real containing the duration in numerical Value compatible with the functions of date and time management (DateTimetoStr...).
SecondsCount : Integer containing the past seconds.
DaysCount : Integer containing the past days.
TimeStr : String containing the past hours, minutes and seconds.

Example :



Vars changing component

Enables to detect that at least a global variable has changed state.
This or these variables are contained in a series of consecutive variables declared in the component by the first variable and the last variable (not included).



Firstvar : Name of the first variable in the group to be watched.
Lastvar : Last not **INCLUDED** variable serving as heel.
It is the variable that is directly declared after the last scanning.
Pulsechange : Boolean fleeting set when one of the variables watches changes. Pulse on 1 cycle of prog.
PermanentChange : Boolean passing to True when one of the variables watches changes.
Boolean stays True and the user program has to reset it.

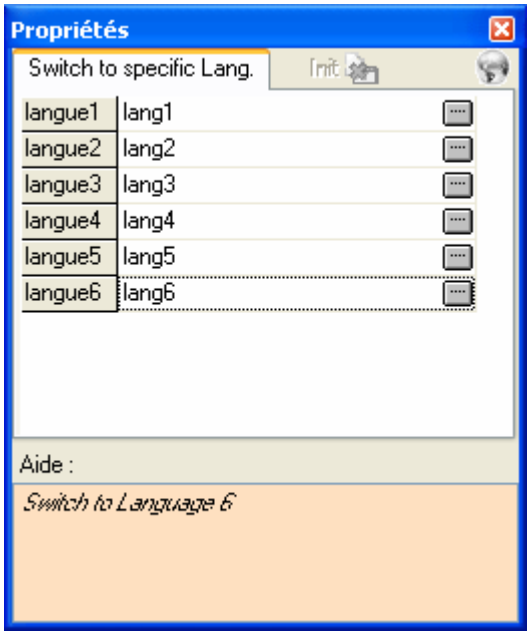
Example of global variables :

```
{******* 8 Bits Words **************}
MB0,MB1,MB2,MB3,MB4,MB5,MB6,MB7,MB8,MB9
{******* 16 Bits Words **************}
MW0,MW1,MW2,MW3,MW4,MW5,MW6,MW7,MW8,MW9                  : Word;
MW10,MW11,MW12,MW13,MW14,MW15,MW16,MW17,MW18,MW19        : Word;
MW20,MW21,MW22,MW23,MW24,MW25,MW26,MW27,MW28,MW29        : Word;  {** Etc.. **}
{******* 32 Bits Words **************}
MD0,MD1,MD2,MD3,MD4,MD5,MD6,MD7,MD8,MD9                  : Integer;
```

In the parameter setting above, only variables in Red are watched. The variable MD0 is not watched. It serves exclusively as a heel.

For example, use this component in the OpenModbus or Modbus settings to start a communication frame on a variable changing state. Use several components to watch different blocks of variables.

Switch to specific language component



Langue1 : Boolean for switching to French language.
Langue2 : Boolean for switching to English language.
Langue3 : Boolean for switching to Italian language.
Langue4 : Boolean for switching to German language.
Langue5 : Boolean for switching to Spanish language.
Langue6 : Boolean for switching to other language.

Other language can be a language unlisted above.
The Unicode characters are used.

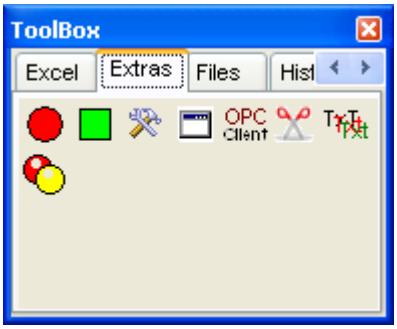
Trends tab

Thanks to peripheral components, this tab enriches the trends viewer that you can find in the advanced tab.
If only one trend is used in your form, all the components placed around it will run with this one.
If several trends curves are used, you need to group the components with the associated trends.

See Creating a page including a trend viewer

Extra tab

This tab groups various components that necessarily have no relation between them.

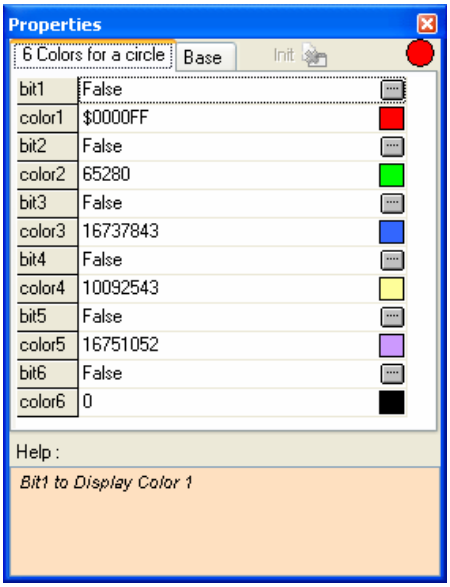


- Fill a circle according to 6 Boolean states
- Fill a rectangle according to 6 Boolean states
- Display a form by shifting.
- Show and hide a form by a Boolean.
- Activation of the OPC client link.
- Detection of the inactivity of the mouse and the keyboard.
- 6 colors for a Text.
- 6 colors for a Led.

Filling a circle according to 6 boolean states.

Filling a rectangle according to 6 boolean states..

Allows testing 6 (Boolean) states and colouring in the circle consequently. If any state is true, the circle is not coloured.

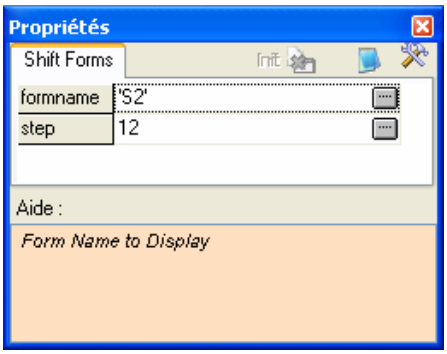


Bit1 to Bit6 : Boolean six (Bits) to be tested. The priority made is from top to bottom (B1 to B6).

Color1 to color 6 : 6 colors corresponding to 6 bits.

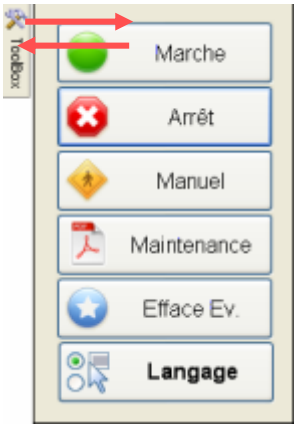
Display and hide a form by sliding.

This component allows sliding out or sliding in a form. The form slides in when the form loses the focus.



FormName : Name of the form to be slided out. The form must be included in the list of the form in the application.

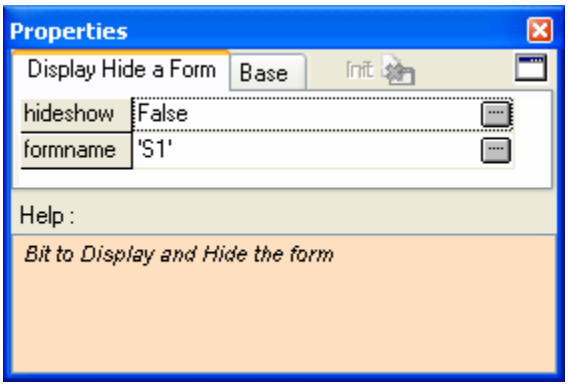
Step : Sliding Speed. The more important this number is, the faster the sliding will be.



Here the form S2 appears by sliding from the left towards the right when the operator spends the mouse on Bitmap ToolBox. The form disappears by sliding towards the left when it loses the focus

Show and hide a form with a bit.

This component allows to show or to hide a form according to the state of a bit. If the bit is true then the form specified is shown otherwise the form is hidden.



Hideshow : Bit to display or to hide the specified form (in formname).
FormName: Name of the form to show or to hide or string variable included the form name.

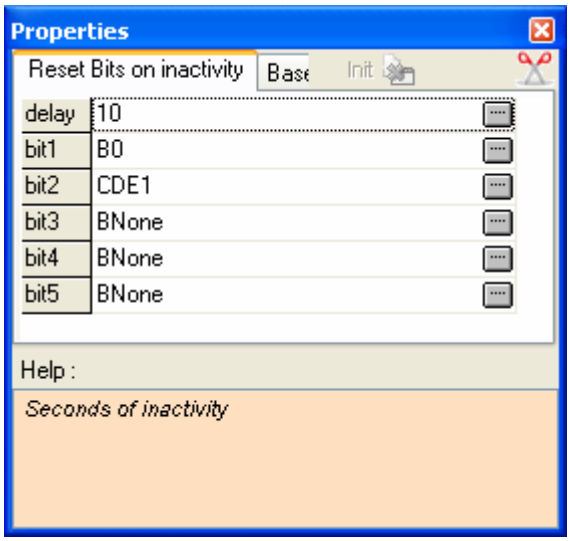
Note: If the name of the form is specified at design time, this one is between two quotes (String constant).

Activation of the OPC client connexion

For reading and writing variables will be done in the OPC server, you must place in the main form this component (for a cyclically operation). Thanks to its property this component enables stop/run the OPC link..
See OPC client chapter

Detection of the inactivity of the mouse and the keyboard.

This component reset (false) 5 bits in case of inactivity of the mouse or the keyboard.
Can use to reset password authorizations .



Delay : Time of inactivity (seconds) before to reset all the bits Bit1 to Bit5.

Bit1 to Bit5 : Bits reset in case of inactivity of the mouse and the keyboard.

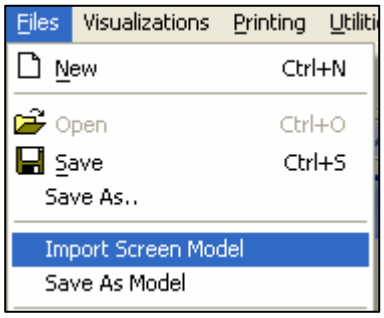
Change color of a text according to 6 boolean states.

Change color of a "Led" according to 6 boolean states..

Allows testing 6 (Boolean) states and colouring in the text or the led consequently. If any state is true, the text color return in the design color.
For a led if any state is true, the color is transparent.
The text has a supplementary attribut and can blink.

Import Export of screens Models

It is possible to save forms as models. Saving is made under directory *Models* of the installation directory.



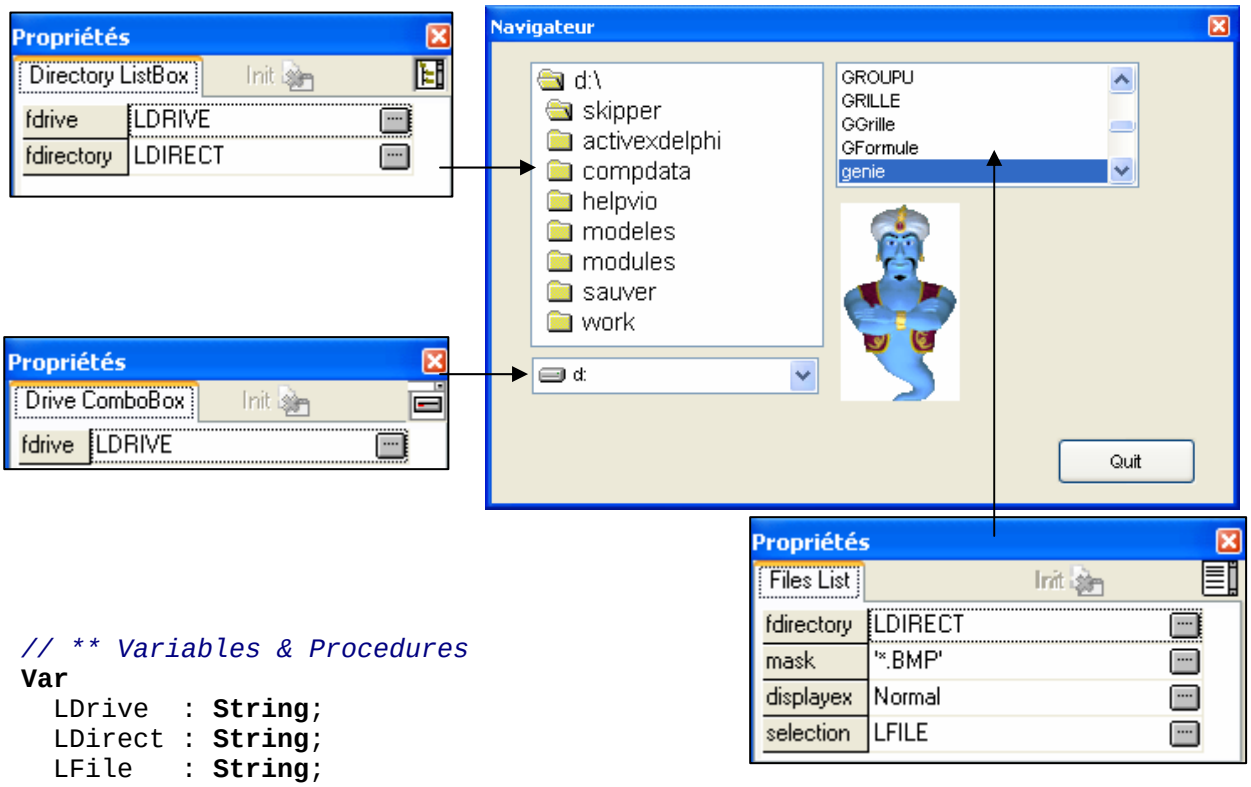
The 2 commands *Import Screen Model* and *save Screen as Model*, allow saving them and reloading them in any project.

These screens are saved with their programs and bitmaps.

It is so easy to create screens to test them and save them as models to be used as base in other projects.

Of course the code source and the graphics are accessible and modifiable in each form imported.
The version 7 of Visual I/O allows using variables declared in the form in the experts.
This functionality is enabled to create models of screens without needing to use global variables

Example Navigator and bitmaps viewer




```
// ** Variables & Procedures
Var
  LDrive  : String;
  LDirect : String;
  LFile   : String;
  Quitter : Boolean;
// *** XPBUTTON1 ******************
SObject Procedure XPBUTTON1;
Begin
  If button then CloseAppli;
End;
```

☞ **Note** : No global variable is used, so a screen is completely relogeable in other applications.

Group

The Group command (the opposite is ungroup) allows associating graphic shapes among them.
This command is valid, when one of the graphic elements is selected under the editor or when a group of elements is selected.
The goal is to maintain graphic spacing among them whatever arrives.

The second function for grouping is to be able to execute an instruction or a complete program to the set of the group.by program

A group of components has a first and a last element. When you group your component, the first of the group is the first drawn in this group. The last one follows the same reasoning.
To modify the order of elements, use ' Bring to Front' command or ' Put Behind '.

The graphic elements in Visual I/O have all the same internal structure.
It is possible to apply the same instructions to them.

Let us take the case of a group containing a circle, a rectangle and a button.

Next_Group

Let us apply from the first element of this group (the circle) the following program:

```
Begin
  Repeat
    SetFillColor(255);    {*** Colouring insides all the drawings in Red **}
  Until Next_Group=False;
End;
```

SetFillColor(255); Color in the inside of a drawing with the specified color (here Red)
The first time, **SetFillColor** is applied to the circle.

Next_Group Go to the following shape in the group (equal to do Go2 instruction to the next drawing).
Next_Group returns False when arriving to the last drawing.

The process of colouring (*SetFillColor*) is applied to the new shape and this, until reaching the last one.

<u>Case or one is not on the first drawing of a group.</u>

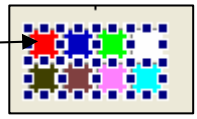
```
Begin
    Repeat Until Previous_Group=False; {*** Go back up to the first shape ***}
    Repeat
      SetFillColor (255); {** Colouring inside drawings in Red **}
    Until Next_Group=False;
End;
```

Previous_Group

Previous_Group Go back up drawings in the group until the first shape.
That makes it possible to start truly on the first shape to perform the treatment

```
 Repeat Until Previous_Group=False;  {*** Go back up to the first shape ***}
```

<u>Example</u> Selecting a color by drag & drop. The first shape (topleft square) contains the following program
The other squres are simply grouped.



```
{*** Shade1 **********************}
SObject Procedure Shade1;
Begin
  {*** Scan le Groupe ****}
  Repeat
    Drag(IntToStr(GetFillColor)); //** Intercepting the mouse
  Until Next_group=False;
End;
```

If a click is performing on one of the 8 squares, the Drag & Drop mecanism is active.

Groupe of objects in a component.

Next_Group_Compo
Previous_Group_Compo

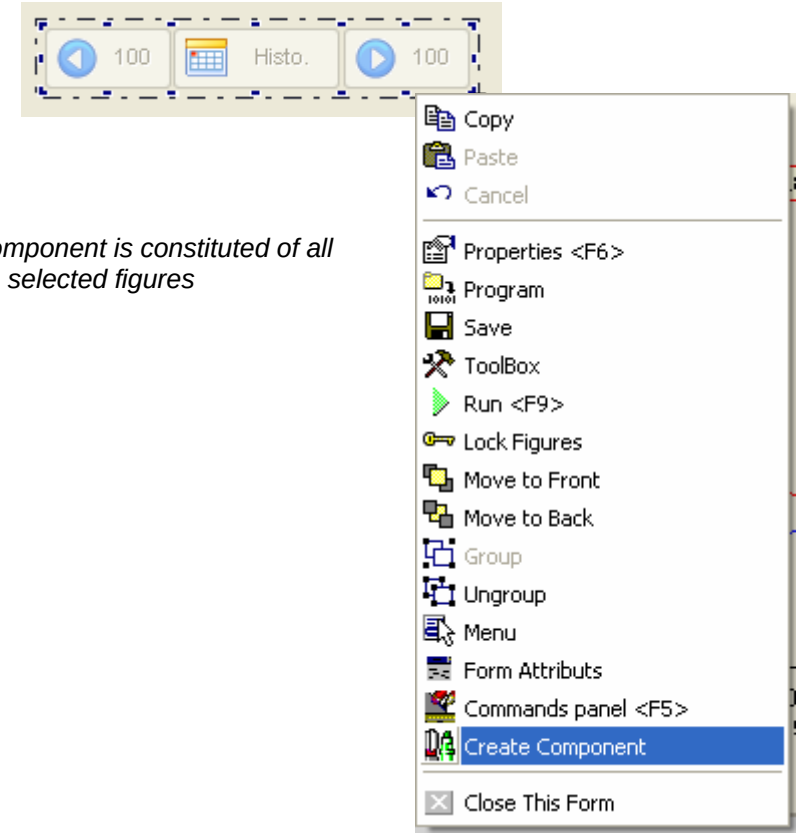
The instructions Next_Group_Compo and Previous_Group_Compo run under the same principle that Next_Group and previous_group scan only the grouped objects of a component..

Example

```
Repeat
Until Next_Group_Compo=False;    //*** Go to the first button of the component
If Button Then
Begin
  …
 Exit;
End;
Previous_group_compo;   //** Go to the second button of the component
If Button Then
Begin
  …
End;
```

The ShHisto component of the Trends tab.

The group of objects of a component is automatically made by selecting all the objects of the component then right clicking with the mouse and selecting " Create component " in the popup menu.

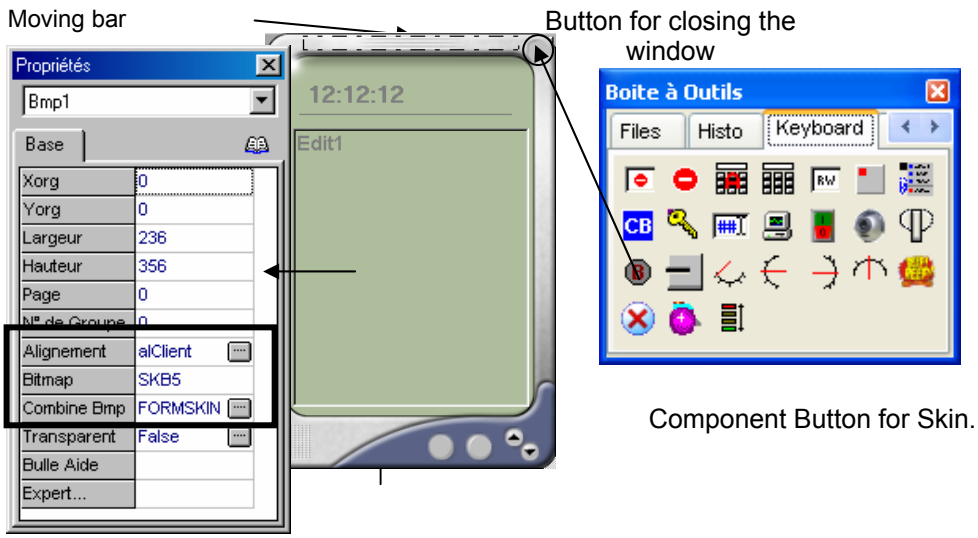


The created component is constituted of all the selected figures

Skins

It is possible to display a form without title bar or borders in Visual I/O.
It is possible for you to design your own contour of window without respecting the Windows' presentation. The form can cover all the possible aspects. A demonstration is visible in the Example of Programming in the Visual I/O installation.

Example



Component Button for Skin.

Directive for use:
At first, place your Bitmap Skin on empty form. Then press key **<F6>** (Call properties box) to select it. Set the **Alignement** property in **alClient** value. Set the property **Combine Bmp** in **FORMSKIN** value.
The bitmap becomes the new form background.

☞ **Note** : to return to the normal aspect Windows, set Combine Bmp with the value Normal.

The moving bar

Select a component Image in the Toolbox to define the contour of the bar for moving the entire Skin.
Place this program behind the image:

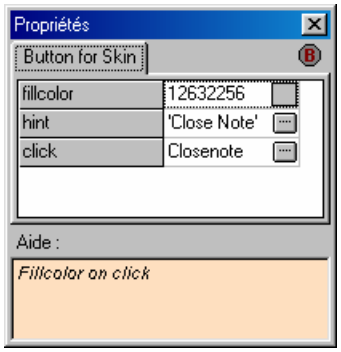
```
SObject Procedure Bmp2;
Begin
  If TopLBDown then Perform(WM_SYSCommand, $f019,0);
End;
```

Place also the directive **Uses Windows** at the beginning of the program.

```
// ** Variables & Procedures *
Uses Windows;
```

This Vpu includes the *Perform* procedure. At runtime the user will be able to use this zone with the mouse for moving the skin (form) replacing the usual title bar.
Component Button For Skin in the keyboard tab

This component can be used for any action by the user. For example, close the form. By double clicking on this button, a property box of appears:



FillColor : Determine the filling color for the circle when the user clicks on it.

Hint : The string to be displayed in the hint window when the mouse cursor passes on this button.

Click : The variable is set (True) when you click on (like a button), the program associated for closing a *Skin synoptic* could be:

```
If CloseNote Then
  Begin
    CloseAppli;   // ** Close the Form *
    CloseNote:=False;  // ** Reset variable *
  End;
```

Translation of forms

Visual I/O allows the multi-languages applications. Thanks to the mechanism of the dictionary, form with static texts, Check boxs, Menus etc., can be translated.
The dictionary is a file text with columns. Each column is separated by a separator that is the character tabulation (# 9).

The first column is French then English, Italian, German, Spanish. The different column is used when one does not use one of the preset languages.
It is very easy to correspond the words in the various languages associated to the corresponding columns.
Use the program delivered Traduct.exe to modify, visualize the Dico.lng file.
Ex : 1 Line of file Dico.lng.

Confirmez #**9** Confirm #**9** Confermare #**9** bestätigen #**9** Confirme #**9** подтвердите
Français Anglais Italien Allemand Espagnol Russe (or other language)

The entire form automatically tries to represent compared to the **Dico.lng** file which is in the directory of your application. If this file is not created and that the language is different from French, your Application.exe creates the file and writes out all the texts of the application.
This file is Excel compatible.
The language at the launching time is the language set in Windows
To change language click on:

The procedure SetLanguage

Procedure **SetLanguage**(NumLang : Integer);
This procedure changes language dynamically. All the texts of your application are retranslated and displayed in the specified language. Numlang varies from 1 to 6 (French to Other).

The Translate function

Function **Translate** (N : String) : String;
This function makes reference to the current language in Windows, compared to the Dico.lng dictionary

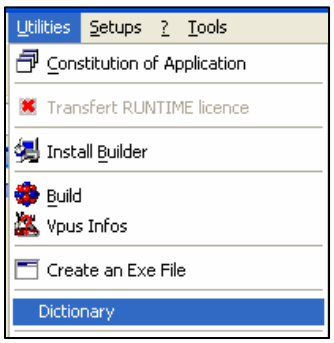
The variable DoTranslate

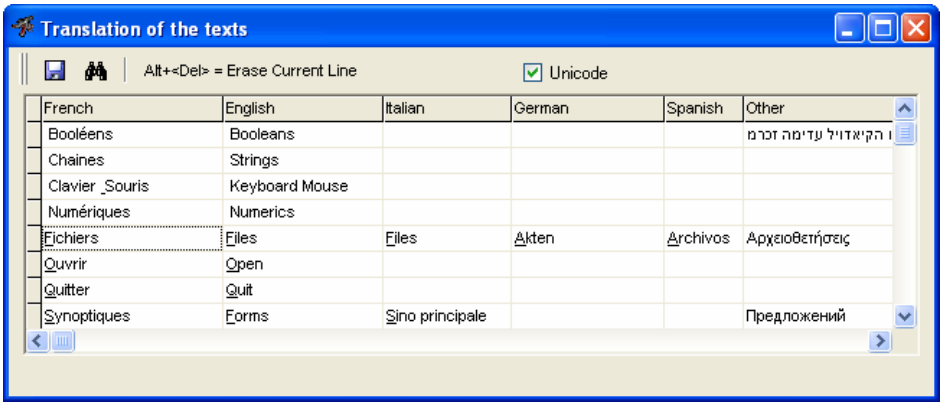
DoTranslate: Boolean=True;
Variable enables or disable the automatic translation.
(False = disable, True =enable (by default)).

If one of these functions should not be translated, insert front the DoTranslate:=false
Example:

```
DoTranslate :=False;
  WinMess(Translate('Open')+NomFichier) ; // *** here the composed string
DoTranslate :=True;
```

Call and modification of the dictionnary





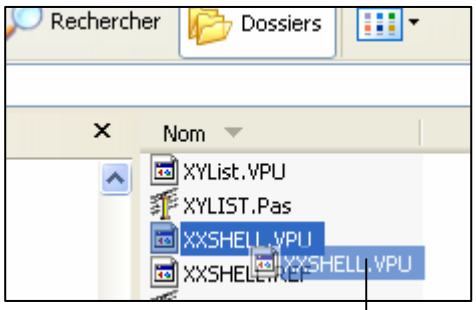
French	English	Italian	German	Spanish	Other
Booléens	Booleans				
Chaines	Strings				
Clavier _Souris	Keyboard Mouse				
Numériques	Numerics				
Fichiers	Files	Files	Akten	Archivos	Αρχειοθετήσεις
Ouvrir	Open				
Quitter	Quit				
Synoptiques	Forms	Sino principale			Предложений

The dictionary is a grid with several columns to return the various texts used in the application in the various languages. The Unicode characters can be used, which authorizes to the display of Arab Cyrillic Chinese etc.
If a language does not have a column, it will be necessary to return the texts in other columns. In exploitation, If the language is not found, the English text is selected.

The Drag & Drop

The basic sequence involved in a drag-and-drop operation is to (1) press, and hold down the button (usually the left button) on the mouse or other pointing device in order to grab the object, (2) drag the object with the pointing device to the desired location on the display screen and (3) drop the object by releasing the mouse button.

The origin : DRAG
The destinations : DROP and GetDrop.
From explorer



Coming from listbox1
Drag(FDirectory);
Defining a possible point of starting for the drag & drop. If the user starts the mechanism from listbox1, the FDirectory string is memorized in order to be transported towards another destination element.

***DragAcceptFiles**(GetHWnd,True); This function behind a component authorizes Drag and drop coming from the Windows' explorer.*

Drag(S : String) ;
Allows storing the character string serving as transfer during an operation of drag and Drop.
The S string is loaded at the time of the mouse left click in an internal character string serving as transport towards an element of reception (Drop). The cursor changes.
If the operator releases the mouse button, the cursor becomes normal again and the internal string is reset.
The internal string is dropped in an element containing the instruction Drop. This element of destination is indicated by the change of cursor aspect in arrow.
Drop : Indicate that the figure is an element of destination of Drag and Drop. When the cursor of the mouse moves through this element, the aspect of the cursor changes. Drop returns true if the mouse is released on the element.

GetDrop : Returns the contents of the string affected by the DRAG function.
DragFrom : Returns the shape name at the start of the Drag & Drop mechanism.

```
{*** ListBox2 ************************************}
SObject Procedure ListBox2;
Var
  TS:String;
Begin
   If Start_visu Then
     DragAcceptFiles(GetHWnd,True); //** Accept files from windows' explorer
  If Drop Then
   Begin
     {*** Receipt Selective Drop ****}
     If DragFrom = ('LISTBOX1') Then Addstring(GetDrop)
                                Else SetFillColor(StrToInt(GetDrop));
   End;
End;
```

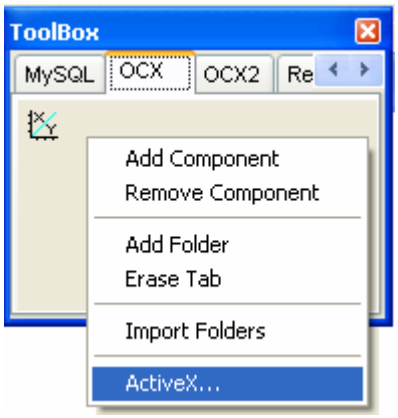
Using an ActiveX

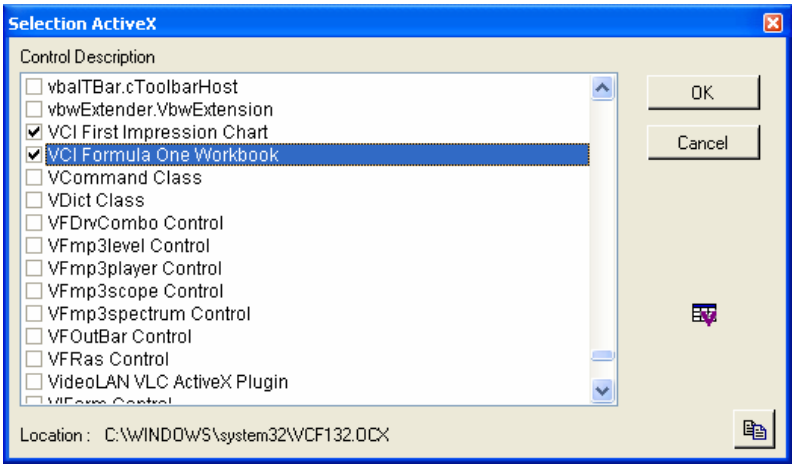
A software module based on Microsoft's Component Object Model (COM) architecture. It enables a program to add functionality by calling ready-made components that blend in and appear as normal parts of the program. They are typically used to add user interface functions, such as 3D toolbars, a notepad, calculator or even a spreadsheet
.

ActiveX controls were originally called "OLE controls or OCX" and used an .OCX file extension

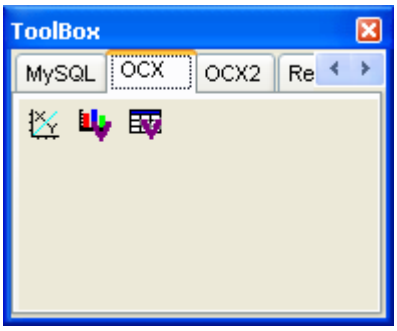
Scan ACTIVEX installed in your PC.

Call the toolbox, create one additional tab (example OCX) then click right with the mouse



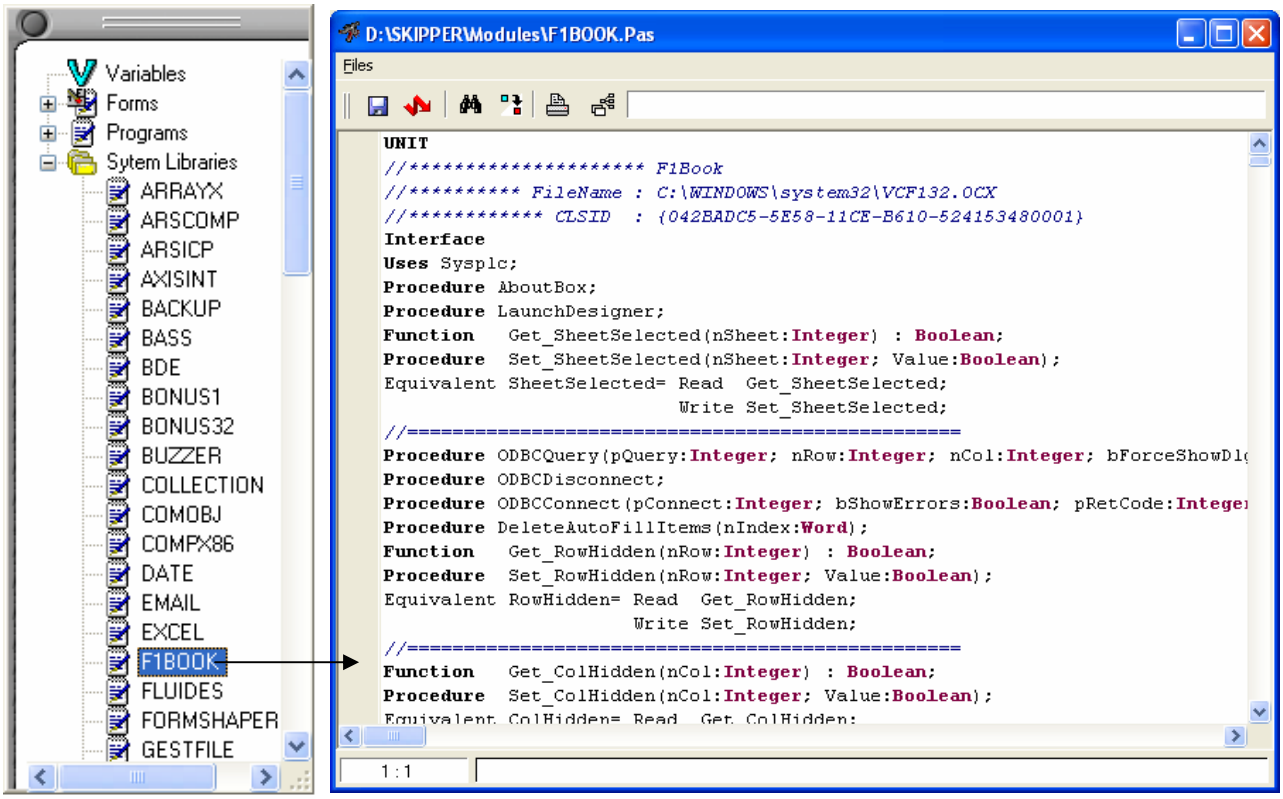


The list of the OCX **installed** in your PC is displayed. Choose simply by notching the check box on the left of the selected ocx.

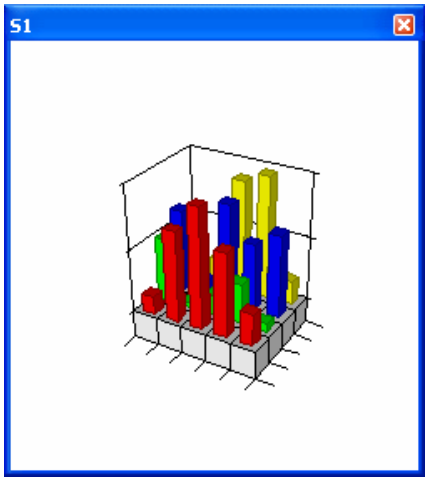


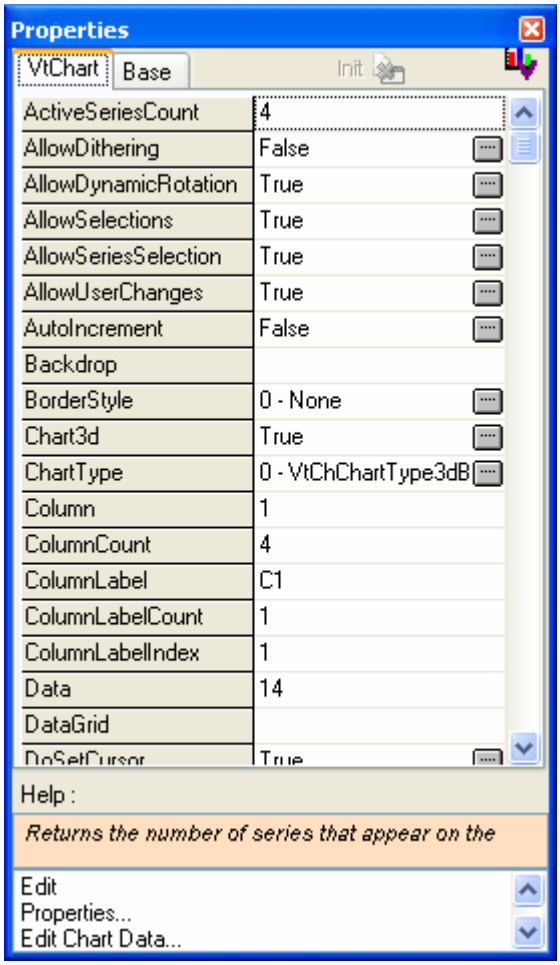
◄ The 2 ActiveX are placed in the toolbox.
You can now use them as a traditional Visual I/O component.

2 VPU libraries are generated automatically to give access to all the procedures and the functions implemented in this ocx.
If the OCX is called **F1BOOK,** the Visual I/O component I/O is called **F1BOOK.VPU**
With ActiveX we enter directly in the advanced programming language of Visual I/O.
Consult the interface concerning the programming, the libraries and the language before starting.



Place VTCHART ActiveX
The ActiveX displays its default settings through the properties box. You can modify these settings.





<F1> call the help provided with the OCX.
Help generally allows documenting the internal procedures of the OCX.

Behind this ActiveX in the interface programs, you will also be able to use the procedures and methods that are included in this ActiveX.

The bottom list is enabled to call the ActiveX internal dialogue boxes by double click.
.

Programming an ActiveX

After having placed your OCX on the form, you will be able to use its procedures and its functions in order to control it with your own way. In VPU file generated automatically at the ActiveX addition time in the toolbox, you will find several types of information.

The constants

Who will be more explicit than simple values. These constants are also generally in references in the ActiveX help.

Example:

```
// *** FontStyleConstants
VtFontStyleBold    = 1;
VtFontStyleItalic  = 2;
VtFontStyleOutline = 4;
// *** FontEffectsConstants
VtFontEffectStrikeThrough = 256;
VtFontEffectUnderline     = 512;
// *** FrameStyleConstants
VtFrameStyleNull = 0;
```

The properties

Who are used as variables in the programs, but in fact, which refer to 2 procedures (one in reading and the other in writing).

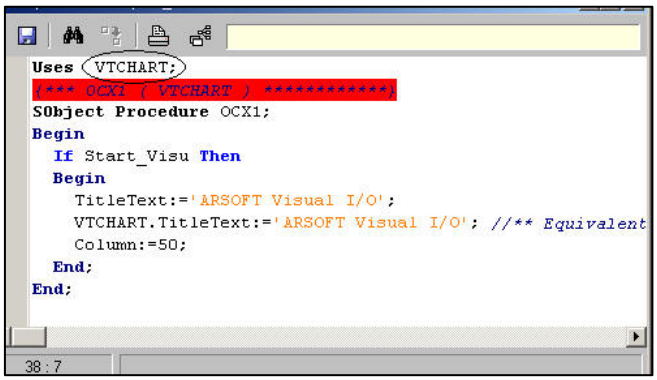
Example:

```
// ==================================================
Function  Get_Column : Word;
Procedure Set_Column(Column:Word);
Equivalent Column= Read Get_Column;
                   Write Set_Column;
// ==================================================
Function  Get_ChartType : Word;
Procedure Set_ChartType(ChartType:Word);
Equivalent ChartType= Read Get_ChartType;
                      Write Set_ChartType;

When the program reads, the GET_xxx function is called
When the program writes, the SET_xxx procedure is called
```

In programme you will write :

```
CPT:= Colum;   // *** Returns the column number call auto of Get_Column
Colum:=12;     // **** Set the column number  call auto of Set_Column
```

Example of program on VTChar



Here TitleText is enabled to set the title of the graph. The directive Uses VTCHART is necessary so that the compiler can find the procedures and variable functions used.

Write **VTCHART**.TitleText is enabled to specify the vpu that contains the procedure TitleText. In this case you will use 2 distinct ActiveXs containing a similar procedure name. (ex: ABOUT).

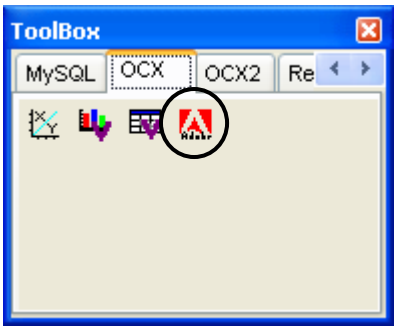
Note:
If a form contains only one ActiveX, the methods (procedures and functions) of this ActiveX can be called everywhere in your program.

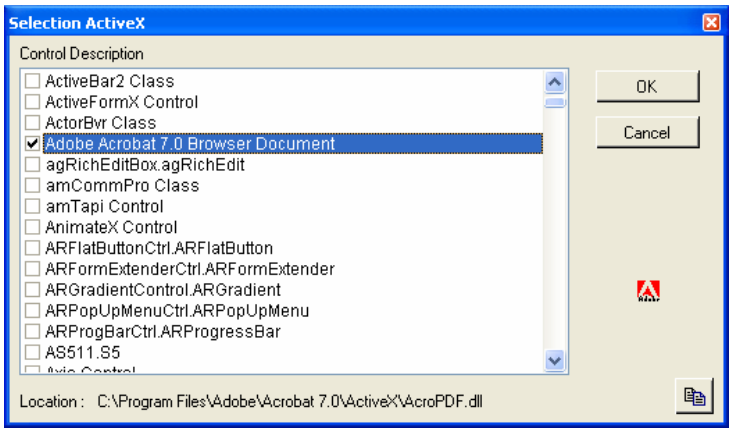
In other case, it is necessary to implement the **Go2** procedure to target the desire ActiveX.

Example:

```
Begin
  Commut_Fig('OCX1');
  About;
End;
```

Acrobat activex from adobe





Select the adobe Acrobat activeX Control to use in your application

Control activeX comes to be placed in your toolbox in the chosen TAB before the scan of ActiveX (see preceding chapter). With this component Visual I/O generates a VPU for interfacing Acrobat. This VPU is named PDF.PAS and PDF.VPU. You will find these in following functions.
It is necessary to contact Adobe to have the detail as of the these functions.

```
Procedure AboutBox; //*** Returns the About box
Procedure setShowScrollbars (On : Boolean);
Procedure printAllFit (shrinkToFit : Boolean);
Procedure printAll;
Procedure printPagesFit (from:Integer; to_:Integer; shrinkToFit:Boolean);
Procedure printPages (from:Integer; to_:Integer);
Function  LoadFile(fileName:String) : Boolean;

Function   Get_src : String;
Procedure  Set_src(src:String);
Equivalent SRC= Read  Get_src;
               Write Set_src;
```

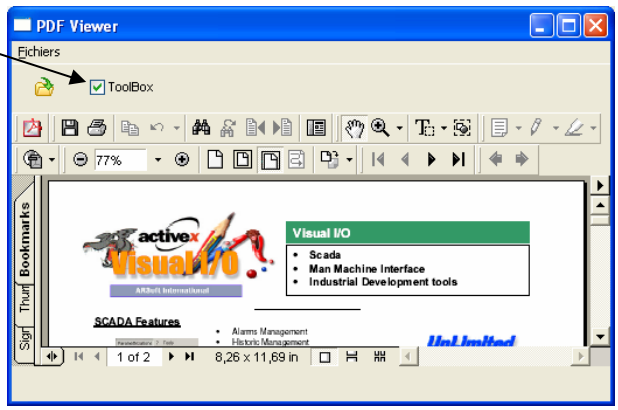
Some functions names are intuitive as Aboutbox that shows a dialog box, printall, whose function is to print the contents of PDF file, or loads File whose function is to load and display a PDF File.
Thanks to the mechanism of equivalence of Visual I/O, the pseudo variable SRC allows reading or changing the name of PDF file to be displayed.

To change the file name to be displayed in the Acrobat OCX, 2 possibilities are allowed:
Uses PDF;

```
If OpenPDF Then PDF.Loadfile(PDFName);  //** Call the procedure LoadFile
Or
If OpenPDF Then PDF.SRC:=PDFName; //** Affect the Variable SRC
```

Here the boolean using **ToolBox** is the state of the checkbox.

```
{*** OCX1 ( PDF ) ************}
SObject Procedure OCX1;
Begin
  If OpenPDF Then
  Begin
     PDF.Loadfile(PDFName);
     PDF.setShowScrollbars(ToolBox);
     PDF.setShowToolbar(ToolBox);
  End;
End;
```



The Variants

A variable in Variant type can take any type. That constitutes a convenient means to handle a variable without knowing its type at the compilation.
ATTENTION, this convenience is paid with the detriment of a slower code. The compiler, with the result that the majority of the errors appear in execution, authorizes all the operations. Thus they should be used with parsimony.

The operations where the variants are used are slower than those bearing on statically associated types. Last, the illegal operations relating to the variable ones frequently cause errors of execution, whereas the same errors with normal variables are detected at the compilation time.

Visual I/O is enabled to work on the variants thanks to the **XVariants.pas** library. The variant is determined like a record containing 16 bytes. In XVariants.pas it is defined like this:

```
Variant = Record
     VType     : Word;  //** Type of Variant
     Reserved1 : Word;
     Reserved2 : Word;
     Reserved3 : Word;
     Value     : Array [1..8] Of Byte; //** Pointer value depending on the type
  End;
```

Vtype can take these values or the combination of these values

```
  varEmpty    = $0000;  varNull     = $0001;   varSmallint = $0002;
  varInteger  = $0003;  varSingle   = $0004;   varDouble   = $0005;
  varCurrency = $0006;  varDate     = $0007;   varOleStr   = $0008;
  varDispatch = $0009;  varError    = $000A;   varBoolean  = $000B;
  varVariant  = $000C;  varUnknown  = $000D;   varByte     = $0011;
  varString   = $0100;  varAny      = $0101;   varTypeMask = $0FFF;
  varArray    = $2000;  varByRef    = $4000;
```

If VType is equal to **VarInteger**, Value[1] to Value[4] contain the integer etc..

Generally, a pointer on a variable is returned because the function returns only registers. In the case of a pointer on Variant, it is necessary to recover the totality of variable pointed on a local variant.
Use the PVar2Var procedure.

Example:

```
Var
 P : Pointer;
 V : Variant;
Begin
  P:=ReturnVar.. //** Function X returning a Pvariant
  PVar2Var(P,V); //** The procedure allows copying  in a variant (V)
                 //** local for treatment.
End;
```

To get a value in a variant use the functions :

```
Function  VariantToBool   ( Var V : Variant) : Boolean;     //** Returns Boolean from Variant
Function  VariantToInt    ( Var V  : Variant) : Integer;    //** Returns Integer from Variant
Function  VariantToWord   ( Var V : Variant) : Word;  //** Returns Word (Smallint) from Variant
Function  VariantToSingle ( Var V : Variant) : Single;     //** Returns Single from Variant
Function  VariantToDouble ( Var V : Variant) : Double;    //** Returns Double from Variant
```

To set a value in a variant use the procedures :
Procedure **VariantFromBool** (Var V : Variant; Value : Boolean); *// Affect an Boolean Value to variant*
Procedure **VariantFromInt** (Var V ; Value :Integer); *//** Affect an integer Value to a variant*
Procedure **VariantFromWord** (Var V ; Value :Integer); *//** Affect a Smallint Value to a variant*
Procedure **VariantFromSingle** (Var V ; Value : Single); *//** Affect a Single Value to a variant*
Procedure **VariantFromDouble** (Var V ; Value : Double); *//** Affect a Double Value to a variant*

Example:

```
Var
  I,J : Integer;
  V   : Variant;
Begin
   VariantClear(V); //** reset the variant V
   I:=123;
   VariantFromInt(V,I); //** Set  the value 123 to V
   J:=VariantToInt(V); //** J becomes equal to 123
End;
```

Variant containing a array

The returned variant can be an array (VarArray). In this case it is necessary to know its dimension, its index of starting and its final index. The functions are available in the XVariants library.

```
//** Indicates whether the specified variant is an array..
Function  VarIsArray (Var A: Variant) : Boolean;

//** Returns number of dimensions of a variant array.
Function VarArrayDimCount  ( Var A: Variant): Integer;

//** Returns the low bound of a dimension in a variant array  Ex : 1 Array [1..?]
Function VarArrayLowBound  ( Var A: Variant; Dim: Integer): Integer;

//** Returns high bound for a dimension in a variant array (Ex : 10   Array [?..10]
Function VarArrayHighBound ( Var A: Variant; Dim: Integer): Integer;

//** Locks a variant array and returns a pointer to the data.
Function VarArrayLock   ( Var A: Variant): Integer;

//** Unlocks a variant array
Procedure VarArrayUnlock ( Var A: Variant);
```

Example: If one knows the size and the type of the data in the array, an array of 10 integers in the variant is to be recovered in a local array (Tablo).

```
Var
  Tablo : Array [0..9] of Integer; //** retrieve  the data of the variant
  P  : Integer;
  PC : Integer;
  V  : Variant;
Begin
   PVar2Var(P,V);
   PC:=VarArrayLock(V); //** Returns a Pointer on the data
   Move(PC^,Tablo[0],Sizeof(Tablo)); //** Perform a Block move of the data
   VarArrayunLock(V); //** Unlock of the data.
End;
```

If we do not know the size of the array or array of variant type has different type to each index,
it is possible to recover an array in a variant where index 1 is an integer, index 2 a Boolean index 3 a double.
In this case, it is necessary to recover index by index and to typecast individually according to its type.

Example:

```
Var
 A,B     : Variant;
 P,I,J   : Integer;
 TB      : Boolean;
 index,LowVal,HighVal : Integer;
 S       : Single;
Begin
  if Button Then
  Begin
     P:=Func..; //** Returns a pointer on the variant
     PVar2Var(P,B); //** Stored in local the variant
     TB:=VarIsArray( B); //** test if it is a array
     If Tb Then
     Begin
        Index   := VarArrayDimCount(B); //*** Array Dimension
        lowVal  := VarArrayLowBound(B, index); //** First Indice
        highVal := VarArrayHighBound(B, index); //** Last Indice
        For J := lowVal to highVal Do //** Scan the Array
        Begin
          VariantArrayGet(B,J,A);
          Case Vartype(A) Of
            varSmallint : I:= VariantToWord(A);
            varInteger  : I:= VariantToInt(A);
            varSingle   : S:= VariantToSingle(A);
          End;
        End;
        VarArrayunLock(B);
     End;
  End;
End;
```

Returns of the function of an ActiveX in PVariant type

Some of ActiveX can return a variant. In this case, the function will turn over a pointer on variable (non-permanent variant) in Visual I/O. Then it should be transferred in a local Visual I/O variable.

<u>Example :</u>
In a VPU the function as follows
Function ReadBinary (maxBytes:Integer) : PVariant;

In a program

```
Uses XVariants;
Var
  V : Variant;
  P : Integer;
Begin
    P:= ReadBinary(100);//** Retrieve the pointer on variant
    PVar2Var(P,V);      //** Move the contain of the variant returned in
                        //** the local variant
End;
```

Programming

It is so difficult to realize an application only using dialog boxes. Visual I/O allows the programming in PASCAL language. Inspired of the Pascal of Borland and from products like Visual Pascal and ML-Visu development tools from ARSOFT International, Visual I/O is based on the scientific language of the Seventies and continues to promote it.

Visual I/O enables the programmer to use an advanced language as well as hundreds (not to say thousands) of procedures and functions to handle all the graphic objects placed in the forms. The Pascal language was named after Blaise Pascal, a French mathematician who was a pioneer in computer development history. In 1641, at the age of eighteen, Pascal constructed the first arithmetical machine, arguably the first computer.

The PASCAL of Visual I/O is a compiled language, i.e. which is necessary:

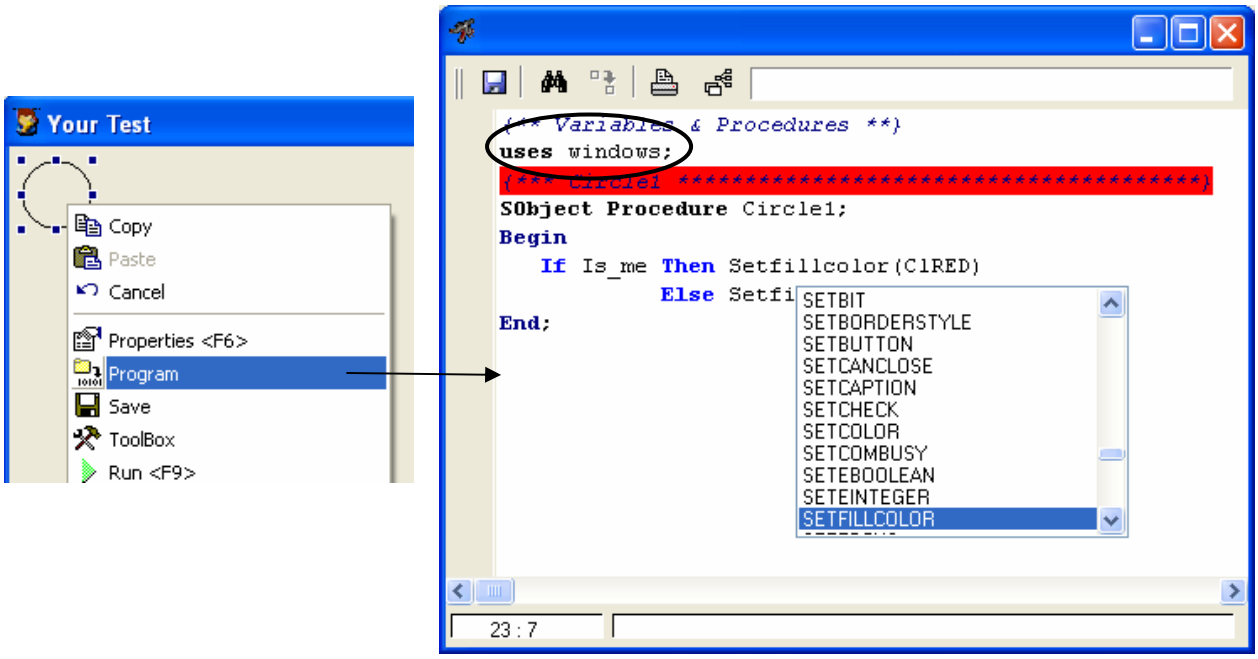
- To enter a text into the computer (using a program called EDITOR),
- To translate a high-level language into machine code (i.e. in comprehensible binary codes by the computer): it is the compilation,
- Then to execute it. Code included in only one file is called executable (with extension EXE)

Contrary to a BASIC (or VB) interpreted, the execution will be much faster since there is no more translation to carry out. Moreover, contrary to other products the executable created by Visual I/O does not require any DLLs to be installed. You can diffuse your application with a simple diskette.

Although the language is standardised, a certain number of points were improved for the supervision applications. Like the test of a bit in a word (W0.0), Micro objects, optional parameters etc. No indications will be given here.

Creation of a program in a form behind a graphic object

Select an unspecified object (for example, a circle), and then click with the right button of the mouse to display the floating menu.



Behind each graphic element, you can manage it using procedures and of functions.
Code completion: An intuitive help allowing for the choice of a list of procedures and functions compared to the text already hit.
<u>The code here:</u>

```
If IS_ME                    // Returns True if the mouse is in the circle
Then SetFillColor(CLRED)    // Color inside the circle in red.
```

CLRED is a constant which is defined in the Library **WINDOWS.VPU**. It is for this reason that declaration **Uses Windows**; exist in the start of the program.

The comments

In Pascal, you can specify a comment in either braces or a slash/slash.

```
{ This is a comment. }
// This is a comment until the end of the line
```

The symbols used to delimit a comment must match.
For example, a comment that starts with { must end with }.

The identifiers

In Pascal, you can include a dollar sign ($) and underscore (_) in an identifier name. The $ and _ can occur in any position of the identifier name. However, you should avoid using these characters in the first position because they may conflict with system names. The compiler can not distinguish between uppercase and lowercase characters.

Here are some useful examples:

CallA TwoNumber ReadNextRecord Call_2

Here are some bad examples:

August {Letter with an accent} 1_Call {Starts with a number}

Call-Sp {the hyphen is forbidden}

Warning : Do not use reserved procedures and functions names defined by the language as identifiers.

The declarations

The declarations include 2 main elements:
The constants (CONST) and the variables (VAR) .
These 2 elements can be written in any order and repeated several times.

Example:

```
CONST
   MaxNum = 1235;
   Chaine    = 'Bonjour';
Var
        I,J   : Integer;    // *** not initialized variable*
        K,L : Word=1;    // *** initialized Variables *
```

Boolean type

In automation of the denomination, Bit corresponds to Boolean under Visual I/O.
However a bit in a word is also included in the Visual I/O Booleans.

The ordinal values of FALSE and TRUE are 0 and 1 respectively.
To initialize a Boolean variable when you declare it in the **Var** declaration of your program, use an assignment statement, as follows:

```
var
        cloudy : Boolean = true;
        sunny  : Boolean = false;
```

Bit in a word

If I0 is Integer type, I0.4 is a Boolean type in Visual I/O (& Visual PLC). Example `I0.4 :=True`

The Array type

Array type (ARRAY) is a structure that includes elements of the same type.
This is the declaration statement of the array:

```
PAIE        : Array [1..4] Of REAL
SEMESTRE    : Array [1..6] Of BOOLEAN
```

Each element is referred by an index. Example Paie[2],

The string type

Strings are arrays of characters. The maximum size of a string type is 255 characters.
The first element of a string (indice=0) contains the length of the string.

This is the declaration statement of the string :

```
S2    : String;
Ich   : STRING ='Manual';  //** Initialized string
```

The index 0 returns the length of the string that is an ASCII value.

A character string is written between two single quotes. If a quote appears in the make up of the character string, a back up copy is taken.

Example:

```
Var
    Msg : STRING;

BEGIN
   Msg : = 'Bonjour';
   Msg :=  'Aujourd''hui' :
END;
```

To know the length of a string, the function Length (Msg) is pre-defined and returns the length of string in reference. In addition, you can also use the ordinal value of the 0 indice belonging to the Ord string (Msg[0]) which returns 7 in the example 'Bonjour'.

If you concatenate a string of 200 characters and another string of 100, the last 45 characters will be lost.

The operators

Operators

Not, /, Mod, And, Shl, Shr +, -, Or, Xor, =, <>, >, <, <=, >=

Some of them will have the weighty task of having to carry out several sorts of different operations.

The arithmetic operators

The arithmetical operators perform the 4 basic operations on numbers; addition, subtraction, multiplication and division, are all represented by their usual symbols in the exception of the multiplication task which is represented by an asterisk *. Another operator is used for dividing: the MOD operator, which oversees the rest of the division operation. Visual IO simplifies arithmetical calculations as far as possible. You can freely mix types. Similarly, the division of a 32-bit byte (Integer) by a Real is possible. The compiler typecast automatically the arithmetical expression when it finds different types in an equation.
If you want to calculate with real constant, you must use number with decimal even if the value is entire
Example R:=W0 * 10.0; where W0 is Integer and R is a real, the calculation is done in real because the constant represent a floating value due to the character . behind 10

Character string operator

The only operating symbol used by the strings is +, perform a concatenation between 2 strings
In each case, the length of strings is limited to 255 characters.

Example:

TS :='Visual PLC'+' Real Time';

The result in TS is : 'Visual PLC Real Time'

To test or affect an element from a character string, use the following syntax.
String[X]

Example:

If TS[1]='v' **Then** Ts[1]:='V';

Logical operators

Logical operators act on the individual bits of ordinal expressions.

- And Bitwise and
- Or Bitwise or
- Xor Bitwise XOR
- Shl Bitwise shift to the left
- Shr Bitwise shift to the right
- Not Bitwise negation (unary)

Here is the truth table for AND OR and XOR :

SHL and SHR carry out the shifting of bits to the right or left of the whole numbers.
Example, I : = 5 SHL 1 gives a result of 10.
The initial byte in binary is 0000 0101, after shifting the byte becomes 0000 1010. The bit furthest most to the left is lost.
On the other hand, if the shifting was SHR 1, the result given would have been: 0000 0010. The bit furthest most to the right is lost.

The Boolean operators

Boolean operators can be considered as logical operations on a type with 1 bit size
Boolean operators can only have Boolean type operands, and the resulting type is always Boolean.
The possible operators are listed below

- And & logical and (The & character is valid as logical operator)
- Or logical or
- Xor logical xor
- Not logical negation

Example:
State:= I0.0 and Cde & Not(W0.12);

Tips & tricks

Affecting the Boolean result

```
Var
     V     : Integer;
     Egal  : Boolean;
Begin
   V:=12 ;
   Egal :=(V=12);     {*** Boolean Result affected to Egal **}
End;
```

The Relational Operators

The relational operators on sets are 4.

Array of all the relational operators combined

= equals.
<> different to.
<= less than or equal to.
>= more than or equal to.

Instructions

The instructions are actions executed inside the block in order of programming.

You can distinguish between simple instructions:

Affectations, procedure and function calls.

And composed instructions:

IF, CASE, REPEAT, WHILE, FOR.

Simple Instructions

The simple instructions are:

- Affectation instructions.
- Procedure and function instructions.

Affectations

The instruction of affectation := enables to load a variable by a new value. The former is an expression of a compatible type with the variable.

Here is some examples:
TS :='Hello' ;
I :=12 ;
R :=12.3 ;

Call of a procedure

Such instruction executes a procedure.

Here is the syntax:
If Button **Then** Initialise;

The number and the type of parameters passed in procedure arguments must correspond to these described in the heading of the procedure called.

The Standard Statements

IF THEN ELSE Statement

In Visual IO the instruction IF THEN ELSE is always used.
The expression between the **IF** and **THEN** keywords must have a Boolean return type. If the expression evaluates to True, the statement following then is executed.
First Case

```
If Counter=10 Then Counter:=0; // here the point comma must be placed behind 0
```

If the variable Counter is equal to 10, Counter is reset.
In this case Else is not used, you must write the character ; behind the last instruction.

2nd Case

```
If Counter =10 Then Counter:=0     // not ; behind 0 because Else is present
               Else Counter:= Counter +1; // Last instruction Then ;
```

If the counter variable is equal to 10, Counter takes the value 0. Otherwise, counter increases.
In this case 'not' is at the end of the first line because Else is used in the second line.
Third Case :

```
If Counter =10 Then
Begin
  Counter:=0;
  Ok:=True;
End;  // *** Last instruction Then ; ***
```

If Counter is equal to 10, execution of a series of instructions is included in the **Begin End**; block
In this case the character is present behind End because no Else is used.
Forth case :

```
If Counter =10 Then
Begin             // First Block of instructions
   Counter:=0;
   Ok:=True;
End Else          // not ; behind End because Else is present
Begin             // Second Block of instructions
    Counter:= Counter +1;
    Ok:=False;
End;              // last instruction then  ;  ***
```

If Counter is equal to 10, execution of a series of instructions is included in the **Begin End**; block;
Else execution of another instructions block.

REPEAT UNTIL Statement

The repeat statement is used to execute a statement until a certain condition is reached.
The statement will be executed at least once.
The statements between repeat and until are executed until the Boolean expression is True.

Example:

```
Repeat
   Counter:= Counter+1;
   Ok:=False;
Until Counter =10;      // *** ; needed
```

The next syntax is correct but useless. The Begin and the End are useless.

```
Repeat
   Begin
      Counter:= Counter+1;
      Ok:=False;
    End;
Until Counter:=10;      // *** ; needed
```

The instruction Repeat Until is the neighbour of the instruction While Do.
However, in Repeat Until the block of instructions is executed at least once, because the test is at the end of the block.

WHILE DO Statement

A while statement contains an expression that controls the repeated execution of a singular or compound statement.
The syntax is **WHILE** Condition is True **DO** Something;
The expression is evaluated before the statement is executed, so if the expression is False at the beginning, the statement is not executed.

Example:

```
While Counter <10 Do
Begin
   Counter:= Counter +1;
   Ok:=False;
End;        // *** ; needed
```

CASE OF Statement

If one of the case constant values matches the value of the expression, the statement that follows this constant is executed
After that, the program continues after the final end. If none of the case constants match the expression value, the statement after the else keyword is executed

```
Example :
Begin
   Case Indice of
            1     : Order:=11;
            2     : Order:=22;
            3     : Order:=11;
            100   : Order:=100;
            Else Order:=0;    // *** Else is optional
      End;
End;
```

IF indice=1 then order receive the value 1.

FOR TO /DOWNTO Statement

A for loop is used in case one wants to calculate something a fixed number of times
Statement can be a compound statement. When this statement is encountered, the control variable is initialized with the initial value, and is compared with the final value. What happens next depends on whether to or downto is used
In the case To is used, if the initial value is larger than the final value, Statement will never be executed.
In the case DownTo is used, if the initial value is less than the final value, Statement will never be executed.
The syntax is :

For .. **To/Downto** … **Do**

Example:

```
      For I :=1 to 10 do Somme:=Somme+Tablo[I];
```

This loop goes from 1 to 10.

Example:

```
      For I :=10 DownTo 1 do Somme:=Somme+Tablo[I];
```

This loop goes from 10 to 1.

Procedures and functions

Procedure statements are called subroutines. There are different possibilities for procedure calls. They are necessary for the good comprehension of a program and to their maintenance. Procedures and functions can be declared in the VPU library part and can use other functions in other VPU

Structure

A procedure is a program part that performs a specific action, often based on a set of parameters.
A function is a program part that computes and returns a value.
Procedures and functions are constituted of a heading and a block.
The heading is constituted by reserved word **PROCEDURE** or **FUNCTION**, followed by an identifier, possibly of a list of arguments in parentheses, and for functions the type of the result returned.

Here is the syntax:

Procedure MyProc1(Element : String) ;
Procedure MyProc2(Var Element : String) ;
Function MyFunc (Number : Real) : **Boolean** ;

In the case of the MyProc1 procedure, the variable Element is sent to the procedure. This Variable can be only read.
In the case of the MyProc2 procedure, the variable Element is sent to the procedure. This Variable can be read and written.
In the case of the MyFunc function, the variable Number is sent to the function. This variable can be read but not written.
This function returns a Boolean that can be tested or not by the instruction has called it.

Returns a result from a Function :

A function returns a typed result (Boolean, Real, Etc..).
This result is a variable created by the function itself. This result variable is a reserved word of the Pascal language named **RESULT**.

Example:

```
Function Addition ( V1,V2 : Real) : Real ;
Begin
   Result :=V1+V2 ;
End ;
```

Result: is the result variable of the function. In the case below Total will receive the result of the Addition function in return.
Total :=Addition(10,20) ;

Optional Parameters in procedures and functions

From Version 7 of Visual IO, the Procedures and functions can have optional parameters.

Example:
Considering a new procedure of filling which permits to colorize the current shape or other shapes:

Syntaxes of call possibles :

```
      SetfillColorEx( CLRED);    or
      SetfillColorEx( CLRED,'RECTANG1');

Procedure SetFillColorEx( Color : Integer; Fig : String);
Begin
  If Fig<>'' Then Go2(Fig);   // if the parameter exists, the system goes
                              // on the object
  SetFillColor(Color);        // Call of the procedure of filling for the
                              // current shape
End;
```

Here the parameter FIG is missing, FIG takes the value of an empty string.
Case of an optional Integer parameter this one takes value of 0.
Case of an optional Boolean parameter this one takes value of False.

☞ **Note**: Examples are readable in the library Bonus32.pas

Visual I/O Pascal instructions set

The instructions set incorporates the compound statements described previously.
IF/THEN/ELSE REPEAT/UNTIL FOR/TO/DOWNTO CASE

The basic instructions are:

Arithmetic operations	+ - * /
Logical operations	OR AND XOR

Shl	Left shift of X bits	`I:=J SHL 3;`
Shr	Right shift of X bits	`I:=J SHR 3;`
Not	Invert a Boolean	`I0.0:=Not(I0.1);`
IntToStr	Converts an integer to a string	`TS:=IntToStr(12);`
StrToInt	Converts a string that represents an integer to a number	`I:=StrToInt('12');`
FloatToStr	Converts a floating point value to a string	`TS:=FloatToStr(12.45);`
IntToHex	Returns the hex representation of an integer	`TS:=IntToHex(I,Digits);`
Move	Copies bytes from a source to a destination	*Move(Source,Destination,count);*
Copy	Returns a substring of a string.	`TS:=Copy('Arsoft',3,4);  //*** TS:='soft' **`
Val	Converts a string to a numeric representation.	`Val(TS,I,Ok);// I=Real OK=Integer); if OK=0  I=Correct value`
Pos	Returns the index value of the first character in a specified substring that occurs in a given string.	`I:=Pos('soft',TS);  // ** I=3 if Ts='Arsoft' *`
Delete	Removes a substring from a string.	`Delete(TS,3,4); // ** TS=Arsoft → TS=Ar *`
Insert	Inserts a substring into a string beginning at a specified point.	`Insert(TS,'Visual Plc ',1); // TS='Arsoft' -→ TS='Visual PLC Arsoft'`
Length	Returns the number of characters in a string	`I:=Length('Arsoft');  // ** I=6 *`
Addr,@	Returns a pointer to a specified variable or procedure	`A:=Addr(Compteur); or A:=@(Compteur);`
Chr	Returns the character for a specified ASCII value.	`TS:=Chr(65);  // *** TS='A' **`
Ord	Returns the ordinal value of an ordinal-type expression.	`I:=Ord('A');  // *** I=65 **`
FillChar	Fills contiguous bytes with a specified value. *This function does not perform any range checking.*	`FillChar(Tablo,16,0); // Tablo : array [1..4] of integer The 4 // first elements of the array are fixed to 0`
Cos	Calculates the cosine of an angle.	`R := Cos(Pi);`
Sin	Returns the sine of the angle in radians	`R := Sin(Pi);`
Exp	Returns the exponential of X.	`E := Exp(1.0);`
Int	Returns the integer part of a real number.	`R := Int(123.456);  //123.0`
Frac	Returns the fractional part of a real number.	`R := Frac(123.456); //0.456`
Trunc	Truncates a real number to an integer	
DegToRad	Converts degrees to radians	`R:=Cos(DegToRad(60));`
Abs	Returns an absolute value..	`r := Abs(-2.3);  //  2.3`
ArcTan	Calculates the arctangent of a given number.	`R := ArcTan(Pi);`
Ln	Returns the natural log of a real expression.	`E := Ln(1.0);`
Sqrt	Returns the square root of X.	
Sqr	Returns the square of a number.	
LoWord	Returns the low order 16 bits of an integer.	`W:=Loword(I);`
HiWord	Returns the high-order 16 of an integer.	`W:=Hiword(I);`
Lo	Returns the low order Byte of argument X.	`W:=Lo(I);  // 0..255}`
Hi	Returns the high-order byte of X as an unsigned value.	`W:=Hi(I);  // 0..255}`

Port	In read returns the value of a physical peripheral
	In write affects a value to a physical peripheral
		`V:=Port($2F8); // uses Sysplc;`
		`Port($2F8):=0; // uses Sysplc;`

Mem	In read returns the value of a memory address In write affects a value to a memory address	`V:=Mem($C800,0);  // ** reading of a byte // needs uses Sysplc;` `Mem($C800,0):=255;  // ** Writing a byte  // needs  uses Sysplc;`

Procedure **ClearBlank** Trims leading and trailing spaces and control characters from a string..
*TS :=Clearblank(TS1) ; //** TS receives TS1 without spaces* Function
StrChange (Ch : String) : Boolean; Return True if the string value has changed.
***If** StrChange(Mystr) **Then** Addstring(MyStr);*
Procedure **HardCopy**; *Perform an Hardcopy of the screen on the current printer.*
***If** Button **then** HardCopy;*
Procedure **MoveFig** ; *Enables moving any shape on the screen with the mouse.*

Procedure **PushButton** (State : Boolean ; Key : Word) ; *Enables a program to anime a simple button.*

// Allow converting a value at scale according to its inputs and outputs limits
Function **Scale** (V : real ; InputLowRange,InputHighRange, OutputLowRange,OutputHighRange : Real) : Real ;

Example: R :=Scale (Eana1,0,2048,-10,100) ;
Eana varies from 0 to 2048 and the equivalent output from –10 to 100. R receives the conversion.
☞ **Note** : This function also runs with integers (typecasting automatically by Visual IO).

Interrupt the execution of your application so that it can process the windows message queue.
ProcessMessages cycles the Windows message loop until it is empty, and then returns control to the application.
Procedure **PROCESSMESSAGES** *// Dispatch Windows messages.*
Procedure **Beep** *// Generate a message beep..*
Procedure **PlaySound** (WavFile : String) ; *// Play a specified. WAV file.*

Instructions concerning the simple shapes

(Circles, ellipses, rectangles ...)
// Return on one cycle the state of simple pushbutton or of a shape used as button
Function **Button**: Boolean;
Procedure **ExclusiveButton** ; *// Exclusive selection on grouped buttons*
Procedure **Filled**; *// Fill a shape with the fill color*
Procedure **UnFilled**; *// Make a shape transparent*
Procedure **FirstColors**; *// Restore the original color (design time) of a shape*
Function **GetButton**: Boolean; *// Return the button state*
Function **GetAngle**:Integer; *// Return the angular value of a Polygon*
Function **GetColor**:LongInt; *// Return the contour color of a shape*
Function **GetFillColor**:LongInt; *// Return the fill color of a shape*
Function **GetPenWidth**:Integer;*// Return the contour thickness of a shape*
Function **GetXFin**:Integer; *// Return the right X-coordinate of a shape*
Function **GetXorg**:Integer; *// Return the left X-coordinate of a shape (x origin)*
Function **GetYFin**:Integer; *// Return the top Y-coordinate of a shape*
Function **GetYorg**:Integer; *// Return the bottom Y-coordinate of a shape (y origin)*
Function **IsFilled**:Boolean; *// Indicate if a shape is filled*
Procedure **SetAngle**(Angle:Integer); *// Set the angular value of a polygon*
Procedure **SetButton**(On_Off:Boolean); *// Set the state of a button*
Procedure **SetColor**(Color:LongInt); *// Set the contour color of a shape*
Procedure **SetFillColor**(Color:LongInt); *// Set the fill color of a shape*
Procedure **SetPenWidth**(Width:Integer); *// Set the contour thickness of a shape*
Procedure **SetXFin**(XEnd:Integer); *// Set the right X-coordinate of a shape*
Procedure **SetXorg**(XOrg:Integer); *// Set the left X-coordinate of a shape*
Procedure **SetYFin**(YEnd:Integer); *// Set the top Y-coordinate of a shape*
Procedure **SetYorg**(YOrg:Integer); *// Set the bottom Y-coordinate of a shape*
Procedure **FloodFill** (Flood : boolean; Color : Integer ; Percent : Integer) ;
// Fill a polygon
Flood = True, the shape is filled. If False, shape is empty.
Color = Filling color
Percent = Percentage of filling
// Allow creating a toggle ebutton with any shape
Procedure **ToggleButton** (Var State : Boolean ; Key : Integer) ;
Procedure **ValidButton** (State : Boolean) ; *// Enable / disable a button*
// Transform a shape into a vertical potentiometer
Procedure **SliderVertical** (Var Variable : Real ;PixelMax : Integer ; LowRange, HighRange : Real) ;
// Transform a shape into a horizontal potentiometer
Procedure **SliderHorizontal** (Var Variable : Real ;PixelMax : Integer ; LowRange, HighRange : Real) ;

Instructions Concerning the ListBox and the ComboBox

Function **AddString** (S:String): Integer; *// Add a string to the end of the List box or ComboBox*
Procedure **ClearList**: *// Erase the contents of List Box or ComboBox*
Procedure **ClearModify**: *// Reset the internal modification flag of an EditBox*
Procedure **ComboHideList**: *// Hide the drop-down list of ComboBox*
Function **ComboSetEditSel**(Var StartP,EndP:Integer):Boolean; *// Select the text in the edit of ComboBox*
Procedure **ComboShowList**; *// Display the drop-down list of ComboBox*
Function **DeleteLine**(LineNumber:Integer):Boolean; *// Remove a line in ListBox or ComboBox*
Function **DeleteSelection**:Boolean; *// Delete the characters selected in a Multiline EditBox*
Function **DeleteString** (LineNumber:Integer):Integer; *// Delete a line in ComboBox or a listBox*
Function **GetCount**:Integer; *// Return the number of elements in a listBox or a ComboBox*
Function **GetHwnd**:Integer ; *// Return Handle of the window containing this instruction*
Function **GetSelCount**:Integer; *// Return the number of selection in a simple or multiselect listBox*

// inform of the state of an element in simple or multiselect listBox
Function **GetSelected** (Item:Integer):Boolean;

// Return the index of the selected line in a listBox or ComboBox
Function **GetSelindex**:Integer;

// Return the selected string in a listBox or ComboBox
Function **GetSelString**:String; .

// Return the contents of the selected element in ListBox or ComboBox .
Function **GetString** (LineNumber:Integer):String;

Function **GetTextLen**:Integer; *// Return the number of characters contained in an editbox or a combobox*
Function **GetValue**:Real; *// Return the value displayed on screen*
.
// allow inserting string into a ListBox or a ComboBox at the given position.
Function **InsertString**(S:String,Position:Integer):Integer;

Function **IsModified**:Boolean; *// Indicate whether the contents of editbox is modified*
Procedure **LDir**(Mask:String); *// perform a directory list in a Editbox or a Combobox*
Procedure **LDirEx**(Mask:String); *// Equivalent to Ldir Plus displaying extension of the files*

Function **SetSelIndex**(LineSelected:Integer):Integer; *// Set the index number in a listBox or a ComboBox*

// Selection of a substring in an editbox beginning at Pos
Function **SetSelString** (S:String,Pos:Integer):Integer;

Procedure **SetText**(S:String); *// Set a text in EditBox or in a ComboBox edit*
Procedure **SetValue**(Value:Real); *// Set the value displayed on screen (object value, editbox)*

// Indicate if the specified string exists in the listBox or the ComboBox.
Function **StrListExist**(String:S): boolean;

// Return the name of the bitmap displayed in the custom ListBox or the ComboBox
Function **VisuBmp**(NumLine:S):String;

Instructions concerning the Checkbox and the Radio Button

Procedure **Check**; *// Force the checkbox into the checked state*
Procedure **UnCheck** ; *// Force the checkbox into the unchecked state.*
Procedure **Exclusive**; *// Allow fixing an exclusive selection on grouped Checkboxes.*
Function **Get_Focus** : Boolean; *// Indicate whether the Windows control has the focus*
Function **GetCheck** : Integer; *// Return the state of a Checkbox 0/1.*
Procedure **SetCheck** (Flag:Integer); *// Set the Checkbox state .*
Procedure **SetRadio**(State : Boolean) ; *// Ses the Radio button state.*
Function **GetRadio** : Boolean; *// Return the state of a radio button*
Function **IsModified**:Boolean; *// // Indicate whether the control is modified*
Function **GetCheckBox** : Boolean; *// Return the state of a checkbox*
Procedure **SetCheckBox** (state : Boolean); *// Set the state of a checkbox*

Instructions concerning the Editbox

```
Procedure Clear;                // Clear the contents of the edit box or the edit of the combo box.
Procedure ClearModify;          // Reset the internal flag of an edit control

// Delete the text in the line specified by LineNumber in multiline EditBox, ListeBox and ComboBox
Function DeleteLine (LineNumber:Integer):Boolean;

Function DeleteSelection:Boolean;  // Clear the selected text in a Multilines edit control
// Return True when user presses < Enter > in an edit control or in an edit of ComboBox (Type 1 or 2).
Function EditCR:Boolean;

// Return True when user presses < Esc > in an edit control or in an edit of ComboBox (Type 1 or 2).
Function EditESC:Boolean;

// Return the contents of an specific line in multiple lines edit control
Function GetLine(Var S:String,Line:Integer):Boolean;

// Return the line from a specific position.
Function GetLineFromPos (CharPos: Integer): Integer;

// Return the length of a line in multiple lines edit control
Function GetLineLength(LineNumber:Integer):Integer;

// Return the number of lines in Multiple lines edit control.
Function GetNumLines:Integer;

// Add a string to a ListBox, ComBox, multiple lines edit control or a grid
Function AddString(S:String):Integer;

// Return the selection bounds in an edit control.
Procedure GetSelection(Var StartPosition,EndPosition : Integer);

// Return the contents of an Edit box or a Combobox.
Function GetText: String;

// Return the length of the text in an Edit box or a Combo box.
Function GetTextLen:Integer;

Function    GetValue:Real;          // Return the value displayed or the value of an edit control.
Procedure Editinsert(S:String);     // insert a substring in an editbox.

// Insert a string at a specific position to a Listbox or a comboxbox. First line is zero
 Function InsertString(S:String,Position:Integer) : Integer;

// True if an edit control, CheckBox, Radiobutton or combobox is modified.
// Calling this function, reset this internal flag.
Function IsModified : Boolean;

Procedure UnCheck ;                 // Uncheck a checkbox.

// Determine whether the user can change the text of the edit control
Procedure SetReadOnly (Readonly:Boolean);
// specify a selection that begins at StartPosition and ends at EndPosition in edit box.
Function SetSelection(StartPosition,EndPosition:Integer):Boolean;

// Set the text string associated with the control Edit box or Combo box.
Procedure SetText(Text:String);

Procedure SetValue(Value:Real);     //  Set the value displayed or the value in an editbox.
Function    LooseFocus : Boolean ;  //  Return true if the control looses the focus
Procedure NextFocus ;               // Skip to the next Windows' control (editbox, checkbox, radio etc..)
```

System Instructions (Keyboard, Mouse, Program ...)

Procedure **ActiveDVE**; *// Activate the forms type Dialog during one cycle*
Function **WinApparition**:Boolean ; *// True (pulse) if the forms appear*
Function **Disparition** : Boolean ; *// True (pulse) if the forms disappear*
// Determine whether the control has a single line border around the client area
Procedure **SetBorderStyle** (Border : Word) ;
Function **GetParentName** : String; *// Indicate the parent name of the drawing*
// Activate the size of the current form (Obsolète now including in the editor)
Procedure RestrictWinSize (WidthMin, HeightMin, WidthMax, HeightMax : Integer) ;

Procedure WinBKGNDColor (Color : Integer) ; *// Determine the background color of the current Form*
// Allow the refreshing of the current form
Procedure **EnableRefresh** (Refresh: Boolean) ;
Procedure **AllIconic**; *// Display all windows as icons.*
Procedure **SetCanClose** (**CanClose** : Boolean); *// Enable/Disable the closing of the form*

Variable **AltActived : Boolean;** *// Reflect the state of the ALT Key.*
Function **Arrive**:Boolean; *// Determine whether the mouse arrive in the component.*
Function **BlinkN**:Boolean; *// Return true every 500 ms.*
Function **BT100**:Byte; *// Return 1 or 0 every 100 ms.*
Function **BT_500**:Byte; *// Return 1 or 0 every 500 ms..*
Procedure **CloseAppli**; *// Close the Application or the current Form.*
Function **Commut_Fig** (DrawingName:String):Boolean; *// Go to a new component.*
Function **GO2**(DrawingName:String):Boolean; *// Go to a new control. Equivalent at commut_Fig*

Function **ControlVisible**: *// Determine if the component is present on screen.*
Variable **CtrlActived** *// Reflect the state of the ALT Ctrl.*
Procedure **CursorNormal**; *// Display the cursor in normal aspect*
Procedure **CursorWait**; *// Display the cursor in wait aspect.*

Variable **DBClick** *// Determine whether a DoubleClick has been performed on the form.*
Function **Depart**:Boolean; *// Determine whether the mouse leaves the component.*
Procedure **Displace**(DX,DY:Integer); *// Move graphically a component relatively.*
Procedure **ODisplace**(DX,DY:Integer); *// Move graphically a component absolutely.*
Procedure **OEnLarge**(DX,DY:Integer); *// Enlarge graphically a component absolutely.*

Procedure **Display**(Name:String); *// Display a form or execute an external program.*
// Display a form or execute an external program and wait for its closing.
Function **FormVisible** (FormName : String) : Boolean; *// Return True if the form is visible on the screen.*

Procedure **DisplayModal**(Name:String);
Function **DoubleClick**:Boolean; *// Determine if a DoubleClick has been performed on the component.*

// Allow storing a string used in a transfer in a drag and Drop operation.
Procedure **Drag**(S:String);

Function **Drop** : Boolean; *// Inform the user of the possibility of drop (by Drag & Drop) in the component*
Variable **End_Visu : Boolean;** *// The last cycle tour of the application before closing.*

Procedure **Execute** (ExeName:String); *// Run an executable.*
Procedure **ModalExecute** (ExeName:String); *// Run an executable and wait for its closing.*

Function **GetCaption** : String; *// Return text string that labels the control.*
Function **DragFrom** : String; *// Return the component name that initiate the Drag & drop mechanism.*
Function **GetDrop** : String; *// Return the internal string of the drag and Drop*

Function **GetHwnd** : Integer; *// Return the current windows handle (window, grid, editbox..)*
Function **GetKey** : Char; *// Return the pressed key*
Function **GetMainHwnd**:Integer; *// Return the handle form*

Function **GetNDecimales**:Byte; *// Return the number of decimals of a numerical value displayed.*
Function **GetPage**:Integer; *// Return the current page in the form(old Visual I/O version)*

// Display a system palette color
Function **GetPaletteColor**(Var Color:Integer):Boolean;
Function **GetTextMessage**:String; *// Return the label displayed on a button or in label*
Function **GetFigName** : String; *// Return the component name*
Function **GetVirKey** : Word; *// Return the Virtual Key pressed.*
Procedure **HideControl**; *// Hide a component*
Function **Is_Me** : Boolean; *// Determine whether the mouse is in the component*
Function **KeyUp** : Boolean; *// Determine whether a key has been released*
Function **LButtonDown**:Boolean; *// Determine whether the left button of the mouse is pressed on the form*
Function **LButtonUp**:Boolean; *// Determine whether the left button of the mouse is released on the form*
Function **LoCaseS** (S:String):String; *// Convert an ASCII string to lowercase.*
Function **UpCaseS**(S:String):String; *// Convert a string to upper case.*
Procedure **SetNDecimales** (Decimales : Integer); *// Determine the number of decimals of a numerical value*
Function **GetNDecimales** : Integer; *// Return the number of decimals of a numerical value*
Procedure **NormalizeTopMosts**; *// Make forms that have been designated as topmost forms behaving as if they were not topmost forms*
Procedure **RestoreTopMosts** ; *// Restore forms designated as NormalizeTopMosts to be topmost again.*
Function **Previous_Group** : Boolean ; *// Go to on the next component*
// indicate whether the right button of the mouse is pressed on the form
Function **RbuttonDown** : Boolean ;
// Indicate whether the right button of the mouse is released on the form
Function **RbuttonUp** : Boolean;
Procedure **SetCaption** (Title : String); *// Specify a label in the title bar of the form*
Procedure **SetPage**(Page : Integer); *// Specify a new page on the form (for old version compatibility)*
Variable **ShiftActived** : Boolean ; *// Return the Shift key state*

Procedure **ShowControl** ; *// Show a component*
Variable **Start_Visu** : Boolean ; *// Indicate the first tour of a form program.*

Function **TopLBDown**:Boolean; *// Indicate whether the left button of the mouse is pressed on the component*
Function **TopLBUp**:Boolean; *// Indicate whether the left button of the mouse is released on the component*
Function **TopRBDown**:Boolean; *// Indicate whether the right button of the mouse is pressed on the component.*
Function **TopRBUP**:Boolean; *// Indicate whether the right button of the mouse is released on the component.*
Function **TypeFig** : Integer; *// Return the type of the shape. See codes in Windows.pas*
Variable **ValidXScroll** (State : Boolean); *// Enable the horizontal scroll bar in the form*
Variable **ValidYScroll** (State : Boolean); *// Enable the vertical scroll bar in the form*

// Specify the text string that can appear when the user moves the mouse over the control
Function **WinMess** (Message:String) : Boolean;
// Return the horizontal coordinate of the mouse relative to its form
Function **Xmouse** : Integer;
// Return the vertical coordinate of the mouse relative to its form
Function **Ymouse** : Integer;
Function **CloseForm**(FormName : String) : Boolean ; *// ** Close the specific form*
Function **FormGetbounds(FormName :** String**; Var Left,Top,Width,Height :** Integer**): boolean;**
*//** Return the Left top coordinates of the specified formname, Width Specifies the horizontal size*
*//** Height Specifies the vertical size Return true if the form exists*
Function **FormSetbounds(FormName :** String**; Left,Top,NewWidth,NewHeight :** Integer**): boolean;**
// Set the Left top coordinates of the specified formname. Width Specifies the horizontal size. Height Specifies the vertical size. If NewWidth or NewHeight equal 0, the width or the height is not modify

Instructions concerning Bitmap

// Animate a simple button with 2 bitmaps
Function **ButtonOwner** (BmpUp,BmpDown:String) : Boolean;

// Animate a button with 2 bitmaps toggle button type
Function **ToggleOwner**(BmpUp,BmpDown:String):

// Specify a text string on a button or a label component
Procedure **SetTextMessage** (BmpFile:String);
Procedure **StretchBmp**(State:LongInt); *// Specify the display attribute of a bitmap.*
Procedure **SetHandleBMP** (Hdle : Integer) ; *// Modify the handle of a bitmap*

// Return the handle of a Bitmap and its Width, Height and palette
Function **OpenBMP**(BmpName : String ; Var Width : Integer ; Var Height : Integer ; Var Palette : Integer) : Integer;

Instructions concerning Menus

Function **GetMenuSt** (id : Word) :String; *// Return the state of a menu item (Checked or not)*
Variable **MenuId : Integer;** *// Return the Id of the menu item clicked.*
Function **MenuStr**:String; *// Return the text string of the menu item clicked.*

Procedure **SetMenuString** (Id : Word; Value : String); *// Specify the text string of menu item (Id)*
// Specify the text string of the element of a menu (Main menu)
Procedure **SetPopupString** (Id : Word; Value : String);

// Specify the text string of the element of a menu (popup menu)
Procedure **SetPopupPString** (Id : Word; Value : String);
Procedure **ToggleMenu** (IDitem : Word); *// Toggle the state of a element menu*
Procedure **ValidMenuItem**(Ident:Word; State:Boolean); *// Enable or disable a menu element*
Function **GetMenuSt** (id : Word) :String; *// Return the state of a specific element*
// Specify a little bitmap (16x16) behind a Menu element
Procedure **SetMenuBitmap** (id : Word ; NomBmp:String);

Others and specifics

Function **GetTrackBar** : Integer ; *// Return the current position of the slider of a TTrackBar.*
Procedure **SetTrackBar** (Value : Integer ; *// Set Position to move the slider of the track bar to a value programmatically.*
// Specify the position of the start & the end point of the selection range
Procedure **SetTrackEND** (Position : Integer) ;
Procedure **SetTrackSTART** (Position : Integer) ; .

Function **GetUPDown** : Integer *// Return the current value*
Procedure **SetUPDown** (Value : integer) ; *// Set the current value*

Display a ladder in a window control

// Specify a Ladder file name to Window control
Procedure **WinLadder** (LadderName : String) ; ***// ** Don't execute in infinitive loop ! ! ! ! ****
Procedure **WinLadderRefresh** ; *// Refresh the Window control containing the ladder*
Procedure **WinLadderZoomIn** ; *// Perform a zoom In on the displayed ladder*
Procedure **WinLadderZoomOut** ; *// Perform a zoom Out on the displayed ladder*
Procedure **WinLadderClear** ; *// Clear the current ladder visualisation.*
☞ **Note** : Use the components in the toolbox

Example of programming behind a Window control.

```
If Start_Visu Then WinLadder('C:\MonProj\LAD1.LAD') ; // ** Load one time *
WinLadderRefresh ;      // ** refresh all the time *
If ZI Then WinLadderZoomIn Else     // ** test if zoom *
If ZO Then WinLadderZoomOut;
```

Display a Grafcet in a Window control

// Specify a Grafcet file name to Window control
Procedure **WinGrafcet** (LadderName : String) ; ***// ** Don't execute in loop ! ! ! ! ****
Procedure **WinGrafcetRefresh** ; *// Refresh the Window control containing the Grafcet*
Procedure **WinGrafcetZoomIn** ; *// Perform a zoom In on the displayed Grafcet*
Procedure **WinGrafcetZoomOut** ; *// Perform a zoom Out on the displayed Grafcet*
Procedure **WinGrafcetClear** ; *// Clear the current ladder visualization*

☞ **Note** : Use the components in the toolbox

Manipulating the coordinates of a form

Function **GetClientOrigin** (Var Xo,Yo : Integer) : **Integer**;
Returns the screen coordinates of the top-left corner of the current form.

Function **GetFormHandle** (FormName : String) : **Integer**;
Returns the form handle of the FormName.

Function **FormSetbounds** (FormName: String; ALeft, ATop, AWidth, AHeight: Integer): **boolean**;
Sets the Left, Top, Width, and Height properties all at once.
Specify the values for the Left, Top, Width, and Height properties as the value of the ALeft, ATop, AWidth, and AHeight parameters, respectively.

Example positionning a form under a component:
Uses Graphics, Windows;

```
 If Button Then
 Begin
   Getclientorigin(Xo,Yo); //** Returns Top-left corner of the component
   PtX:=Xo;
   PtY:=Yo+(GetYfin - GetYorg);      //*** Bottom-right conner
   W:=GetSystemMetrics(SM_CXSCREEN); //** Screen resolution (Width)
   H:=GetSystemMetrics(SM_CYSCREEN); //** Screen resolution (Height)
   FormGetBounds('Mykeyboard',BX,BY,BW,BH); //** Retrieves the form bounds
   IF (Pty+BH)>H then Pty:=Pty-BH-(GetYfin-GetYorg);
   IF (Ptx)<0 Then Ptx:=0;
   IF (Pty)<0 Then Pty:=0;
   //** Display the form below the component thanks its new coordinates
   FormSetbounds('Mykeyboard',Pt.X,Pt.Y);
End;
```

Visual I/O System Variables

DRAGFROM : String ; // Component initiating the Drag & Drop
DBCLICK : Boolean ; // Double click performed on the component
LBUTTONUP : Boolean ; // Release of the left button of the mouse
LBUTTONDOWN: Boolean ; // A left click with the mouse is performed
RBUTTONUP : Boolean ; // Release of the right button of the mouse
RBUTTONDOWN: Boolean ; // A right click with the mouse is performed
BT_500 : Byte ; // Equal to 1 every 500 ms
BT100 : Byte ; // Equal to 1 every 100 ms
CDATE : Real ; // Current Date encoded
CTIME : Real ; // Current Time encoded
SCDATE : String ; // Current date (text string)
SCTIME : String ; // Current time (text string)
CYEAR : Word ; // Current Year
CMONTH : Integer ; // Current Month
CDAY : Integer ; // Current Day
CHOUR : Integer ; // Current Hour
CMIN : Integer ; // Current Minute
CSEC : Integer ; // Current Second
CHSEC : Integer ; // Current 1/10 of second
START_VISU : Boolean; // Return true ONLY on the first apparition of a form
END_VISU : Boolean; // Return true ONLY on closing the application
WINAPPARITION: Boolean ; // Return True every apparition of the form.
WINDISPARITION: Boolean ; // Return True every disappearing of the form.
MENUID : Word ; // identifier of the clicked element in a menu
XMOUSE : Integer ; // Current horizontal coordinate of the mouse
YMOUSE ; Integer ; // Current vertical coordinate of the mouse
XPRINTPAGE : Integer ; // Horizontal Pixel resolution of the current printer
YPRINTPAGE : Integer ; // Vertical Pixel resolution of the current printer
ALTACTIVED : Boolean ; // ALT key pressed
SHIFTACTIVED: Boolean ; // One of the 2 SHIFT key is pressed
CTRLACTIVED: Boolean ; // CTRL key pressed
SCANFORMS : Integer ; // Scan time of the Forms. Default setting =10 ms Mini = 1ms
0=Maxi 10=10 Ms Etc.. (Time of processing the message queue)
PAINTSTRUCT: Integer ; // Pointer on a custom trend structure see chapter

EVENTAPP : Boolean; // Indicate whether a mouse click or a keyboard pressed is performed on the form
DOTRANSLATE: Boolean; // Enable the text string translation
RUNUSERPROG: Pointer; // pointer on a procedure that will be cyclically called by the system (timer)

CALLBACKPRINTALARM : Pointer; // Pointer for a Call back procedure will be called by the alarms manager.
FORMMSGS : Pointer; // Pointer on a custom procedure for windows messages reception

Systems Variables written in bold : New variables in Visual IO version 7.

GLOBVARSIZE : Integer; *// Return the effective size of your global variables*
BASEGLOBVARS : Integer; *// Return the starting address of your global variables*

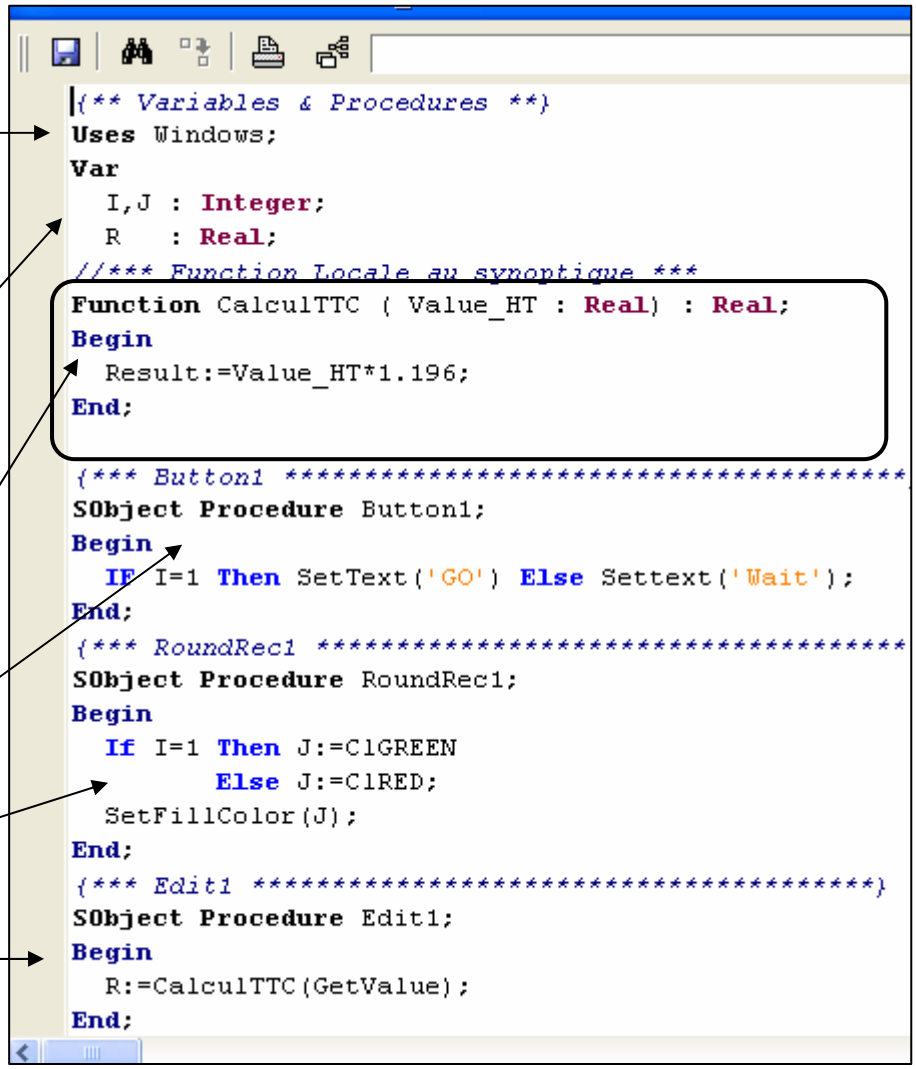
New system variables for general purpose
INT_Sys1 to INT_Sys10 : **`Integer;`**
REAL_Sys1 to REAL_Sys10 : **`Real;`**
BOOL_Sys1 to BOOL_Sys10 : **`Boolean;`**
STR_Sys1 to STR_Sys10 : **`String;`**

Different zones of the program editor in the forms.

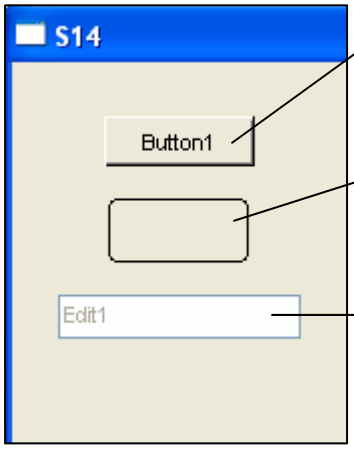
Libraries list necessary to your programs.
Here the procedures, functions and constants of the Windows library are useable in the program of the form.

Declaration of variables locals to the form and common to all the components.
Here 2 variables I and J and a variable R.

A function useable and common to all graphical objects.



```
{** Variables & Procedures **}
Uses Windows;
Var
  I,J : Integer;
  R   : Real;
//*** Function Locale au synoptique ***
Function CalculTTC ( Value_HT : Real) : Real;
Begin
  Result:=Value_HT*1.196;
End;

{*** Button1 ************************************
SObject Procedure Button1;
Begin
  IF I=1 Then SetText('GO') Else Settext('Wait');
End;
{*** RoundRec1 ***********************************
SObject Procedure RoundRec1;
Begin
  If I=1 Then J:=ClGREEN
         Else J:=ClRED;
  SetFillColor(J);
End;
{*** Edit1 ***************************************}
SObject Procedure Edit1;
Begin
  R:=CalculTTC(GetValue);
End;
```



The different portion of code associated to each graphical component.

An internal instruction to colorize **SETFILLCOLOR** can colorize inside the button if this instruction is used between **Begin** and **End** of **SObject Procedure Button1**; or colorize inside the rectangle or the Editbox if this is written in program bloc associated to the component.

The program of the component EDIT1 can use the variable R as well as the procedure CalculTTC.
Getvalue returns the numerical value pressed in the edit box Edit1

From visual I/O version 7, the local variables of a form can be used in the experts.

Before this version 7 the experts could use only the global variables.

The use of locals variables in the experts enables creating relogeables forms in other applications.

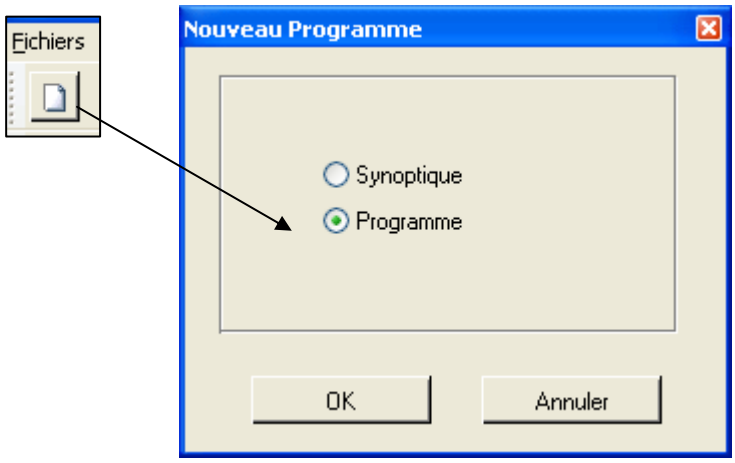
Le associated form programme is in the file name form plus _AUTO.VIO.

Example:

Form	S12.SYN
Programme Source	S12_AUTO.VIO
Compiled Programme	S12_AUTO.VPU

Creation of VPU libraries or cyclic programs

Visual I/O enables creating libraries containing procedures and functions, which you will use in your programs. The libraries are also called Units or more common VPU.



Click on new. A window appears. Check the radio button Program then button OK. A text editor appears which allows creating programs or libraries.

Writing cyclic programs in Pascal language

It is possible to write cyclic programs (VPU) under Visual I/O. Cyclic programs (VPU) will be loaded in the real time engine (Menu utilities/ application constitution).

This editor allows writing libraries as cyclic programs. In the case of a program, **clear** the first line containing the reserved word **UNIT** as well as the line containing the word **INTERFACE**.

A simple program can be:

```
Uses MVarglob;     {** Use the global variables Mvarglob.Vpu **}
Begin              {*** Beginning main program ***}
  D0:=D0+1;
End.         {*** End of main program ***}
```

Autre exemple :

```
Uses MVarglob;  //  Use the global variables Mvarglob.Vpu *
//**** Procedure addition ***********
Procedure CalculPlus( Var XWrd : Integer);
Begin
  XWrd:=XWrd+1;
End;
//*** Procedure convert degrees to radians ******
Function DegreRad ( Xdegre : Real) : Real;
Begin
  Result:= Xdegre *(3.1415926535897932385 /180);
End;

//*** Main *******
Begin            // *** Beginning main program **
  D0:=D0+1;
  Calculplus(D0); //*** equivalent at D0:=D0+1;
  Angle:=DegreRad(Mesure); //** Conversion
  Temperature:=Scale(ValPLC,0,32767,0,100); //**
End.             // *** End of the main body **
```

Note : In this example, it is a program. Procedures and functions used are useable only by the program. If you want to use these procedures and functions in other programs, you must write them in an external library and add its name to the uses clause after MVarglob. ('*Uses MVarglob,xxx;*').

The USES clause

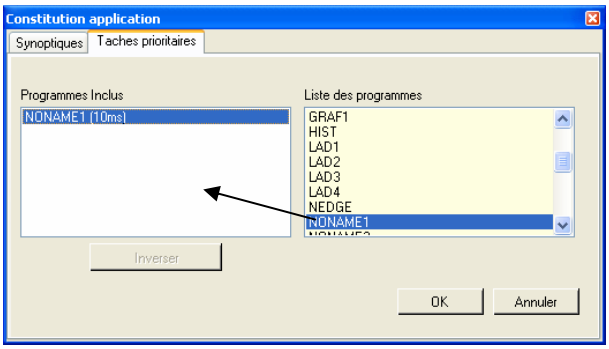
The uses clause is where you list the namespaces that you want to include in a particular program or unit.

For example, if you have a program called FooProg that uses functions and types in two namespaces, UnitA and UnitB, the proper uses declaration is as follows:

Uses UnitA, UnitB;

The main program code

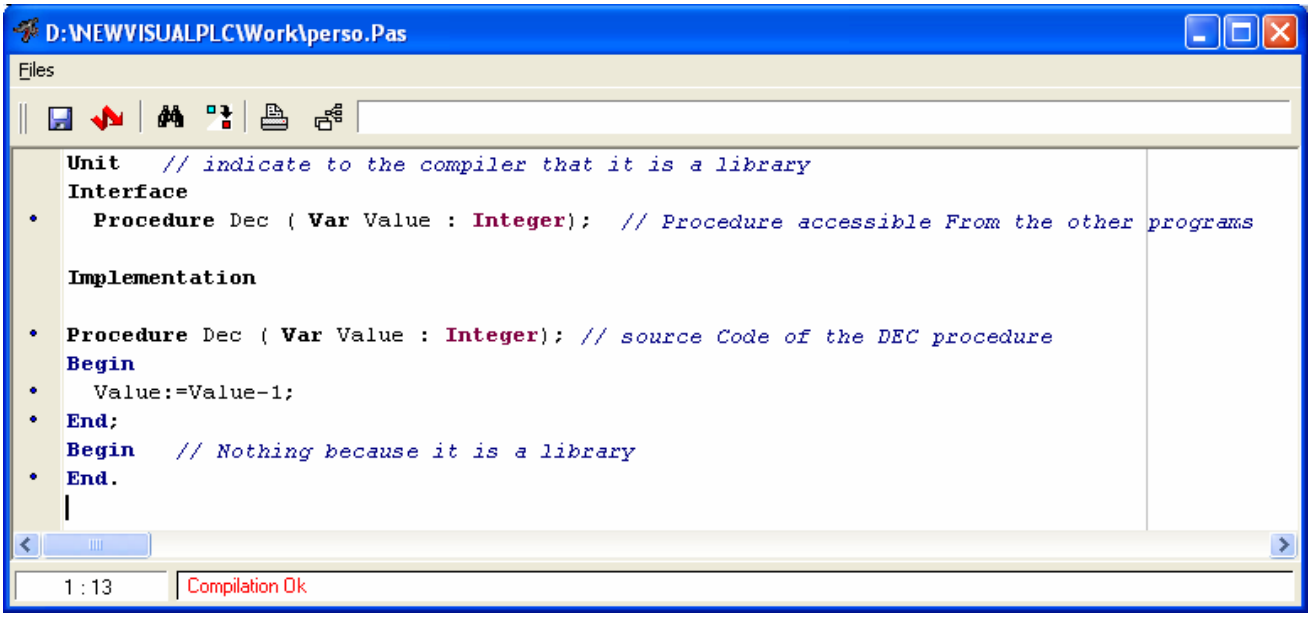
The main program code contains the instructions that must be called cyclically by the priority task in visual I/O. These main programs begin with a **BEGIN** and terminate by an **END.** (Followed by a point and not ;).



Compile, and then load the program compiled with the menu Utilities/Application Constitution in priority tasks.

Writing VPU libraries

The principle seen before remains the same, but the only difference is that a library do not contain cyclic program. A library is passive and export variables, procedures and functions.



.

Let us examine this code:

```
UNIT  // indicate to the compiler that it is a library
Interface
 Procedure Dec ( Var Value : Integer);    // Procedure accessible to the other programs

Implementation

Procedure Dec ( Var Value : Integer);   // source Code of the DEC procedure
Begin
  Value:=Value-1;
End;
Begin // Nothing because it is a library
End.
```

The UNIT clause

The unit header starts with the reserved word unit indicating to the compiler. It is a **library**, not a program. Nothing is executed cyclically in the main between BEGIN and END.

In **IMPLEMENTATION**
The implementation section of a unit is the section that contains the actual code for the unit. The implementation can have additional declarations of its own, although these declarations are not accessible to any other application or unit.
In **INTERFACE**
In the unit interface section, we can declare global constants, data types, variables, procedures and functions.

The DEC procedure : We can see this Value is passed in parameter with the word **VAR** in before.
When a parameter is preceded by the VAR clause, it means that the program in the procedure or the function receiving it can modify it.
In fact, the parameter received by the procedure is not the contents of Value but its address.
Program DEC decreases contents of the variable to the given address.

Initialization in a library

If we want to initialize any data the unit uses, you can add an initialization code to the **BEGIN END** Section of the unit (Main). When an application uses a unit, the code within the unit's initialization part is called before any other application code runs.

Let us examine this code:

```
UNIT          // indicate to the compiler that it is a library
Interface
  Procedure Dec ( Var Value : Integer);   // Procedure  exportable to other programs

Implementation

Procedure Dec ( Var Value : Integer);     //  source Code of the DEC procedure
Begin
  Value:=Value-1;
End;

Procedure Initial;     // Internal procedure in the library
Begin
   …..
 End;

Begin         // Initialization of the library (unit) at the start-up time of the application
  Initial;  // Call initial is done only one time
End.
```

Passing parameters by values and address

When you create procedures and functions, you will have to pass parameters to these procedures and functions. Two solutions are possible.

Passing by value

```
Ex: Procedure Test (Value: Integer);
Begin
End;

Begin
  I:=10;
  Test(I); // ** sent by value to TEST procedure
           // here I is always equal to 10 whatever the action in Test
End.
```

In this procedure, 'I' cannot be modified by the procedure.
We are able to write: If Value = 10 Then … or Counter:=Value;

The expression Value:=100 will not have any effect on the variable 'I'. Affecting Value in the procedure modifies the local value but not the original value. A local copy of I is made in the Test procedure.

Passing by address

```
Ex : Procedure Test( Var Value : Integer);
Begin
   Value:=Value*2;   // *** Modify the value of i *
End;
Begin
  I:=10;
  Test(I);   // ** passing to the procedure Test by address *
             // here I is not equal to 10 it depends on the Test procedure
End
```

When passing data to a routine (function or procedure), you can prefix the parameter definition with Var if the variable itself is to be updated by the routine. This allows a caller to pass data to a routine that will be enriched by the routine.
Let us take the example of the procedure Inc that allows increasing a variable again.

We could write:

```
Procedure INC ( Var Value : Integer);
Begin
  Value:=Value+1;
End;
Begin
  Inc(I); // ** Increase I by 1
End.
```

Or writing a function that will do the action:

```
Function INC (Value : Integer) : Integer;
Begin
  Result:=Value+1;
End;
Begin
  I:=inc(I);
End.
```

Other example:

```
Function Inc(Value : Integer) : Integer;
Begin
  Result:=Value+1;
End;
Begin
  R:=1.2;
  R:=Inc(R);   // ** send a real to the function and retrieve result in a real
End.
```

The INC Function is normally designed to receive and return an Integer. In this case, the compiler transforms R (a real) into an Integer to adapt to the function and transforms the result returned in integer into real.

Passing a variable with no type

We can also write:

```
Procedure Inc ( Var Value);
Begin
   Integer(Value):=Integer(Value)+1;
End;
```

Here the INC procedure receives a non-typed variable and this by address.
It is necessary that the procedure typecasts the variable to exploit it. The INC procedure decides in the example that value is an integer type.

The Visual IO system instructions sorted by categories

The data grid

Visual I/O includes a set of functions to manage grids.
A grid includes a set of data (values or others) in an array. Visual I/O enables several functions allowing the manipulation the data of the grid programmatically. A grid can contain text string, bitmaps and windows controls (List box, Combo Box, Checkbox), lines in different colors.
For creating a grid, there are two possibilities:

- Using the component in the toolbox (Data grid).
- Alternatively, using a program in a window control (Window).

For the different functionalities, see the chapter Data grid.

Case of several grids in a form

Only one grid displayed in the form:
If one grid is present in a form, all functions referring to a grid are sent to it.

Several grids in a form:
It is possible to display several grids in the same form. For applying a function to a grid, it is necessary to call the **Go2** procedure, which means go to the component specified by the name in parameter that creates the grid. Thanks to this procedure, the system knows what the target grid is.

Functionalities oriented Initialization

(Transform the window control into a grid and initialize it).
Procedure **CustomGrid** (Col,Rows, FixedCol,FixedRow : Integer); *// * Rows, columns*

// Specify various display and behavioural properties of the grid
Procedure **GSETOPTIONS** (Options: Integer);

FixedHorzLine	Horizontal lines are drawn to separate the fixed (nonscrolling) rows in the grid
FixedVertLine:	Vertical lines are drawn to separate the scrollable columns in the grid.
HorzLine:	Horizontal lines are drawn to separate the scrollable rows in the grid..
VertLine:	Vertical lines are drawn to separate the scrollable columns in the grid.
RangeSelect:	Users can select ranges of cells at one time.
DrawFocusSelected:	Users can select ranges of cells at one time.
RowSizing:	Scrollable rows can be individually resized.
ColSizing:	Scrollable columns can be individually resized.
RowMoving:	Scrollable rows can be moved with the mouse.
ColMoving:	Scrollable columns can be moved with the mouse.
HideEditing:	Hide the contour of the selected cell.
DisableEdit:	Disable editing in the cells.
Tabs:	Users can navigate through the cells in the grid with Tab and Shift+Tab
AlignCenter, AlignRight	Centers text horizontally and Aligns text to the right

Constant values associated to Windows.pas

FixedHorzLine	= 1;	FixedVertLine	= 2;
HorzLine	= 4;	VertLine	= 8;
RangeSelect	= 16;	DrawFocusSelected	= 32;
RowSizing	= 64;	ColSizing	= 128;
RowMoving	= 256;	ColMoving	= 512;
HideEditing	= 1024;	DisableEdit	= 2048;
Tabs	= 4096;	AlignCenter	= 16384; AlignRight = 32768;

Example: GsetOptions(FixedHorzLine Or FixedVertLine) ; *// Equivalent GSetOptions(3);*
(Send a command to a cell).
Procedure **GCommand** (Tipe : String; Col, Row : Integer);

Tipe : 'COLOR' or 'BITMAP' or 'CONTROL'.
Col : Column number affected by the command.
Row : Row number affected by the command.

☞ **Note** : See chapter on the grids.

*(**Determines the width (in pixels) of a specific column.*
Procedure **GSetColWidth** (Col : Integer; Value : Integer);

(Assign a Window control to a cell).
Procedure **GsetControl** (ACol, ARow: integer; Value : string);

*(**Specify the background color of the fixed rows and columns in the grid).*
Procedure **GSetFixedColor** (C : Integer);

*(**Specify the number of columns on the left of the grid that cannot be scrolled).*
Procedure **GSetFixedCols** (C : Integer);

*(**Specify the number of rows on the top of the grid that cannot be scrolled).*
Procedure **GSetFixedRows** (R : Integer);

*(**Specify the height (in pixels) of all rows in the grid.).*
Procedure **GSetRowHeight** (R : Integer; V : Integer);

*(**Specify the text color in the fixed rows and columns in the grid).*
Procedure **GSetTextFixedColor** (Color : Integer);

(Enable editing in the cell by double click).
Procedure **GSetDoDbClick** (Enable : Boolean);

Functionalities oriented navigation

(Indicate whether the content of the cell has changed).
Function **GCellChange** : Boolean;

(Indicate whether the used changed of cell).
Function **GDeplaceFocus** : Boolean;

Functionalities oriented control

(Indicate whether a doubleClic is perform in the grid).
Function **GDbClick** : Boolean;

(Enable the editing in the current cell).
Procedure **GValidEdit**;

(Disable the editing in the current cell).
Procedure **GDevalidEdit**;

(Enable or disable repainting in the grid).
Procedure **GEnableDisplay**(On_OFF : Boolean);

(Set a Background cell Color).
Procedure **GSetCellColor**(ACol, ARow: Integer; Color : Integer);

*(**go to a specified column).*
Procedure **GSetCol**(V : Integer);

(Specify the number of columns in the grid).
Procedure **GSetColCount**(V : Integer);

*(**Specify the index of the first visible scrollable column in the grid).*
Procedure **GSetLeftCol** (LeftCol : Integer);

*(**Specify the index of the first visible scrollable row in the grid.).*
Procedure **GSetTopRow** (TopRow : Integer);

*(**Specify the index of the row that contains the selected cell).*
Procedure **GSetRow** (V : Integer);

(Specify the number of rows in the grid.).
Procedure **GSetRowCount** (V : Integer);

(Set a Cell Text Color).
Procedure **GsetTextColor** (Colum, Row : Integer ; Color : Integer) ;

Functionalities oriented Information

*//** (Return the row and column cell index pointed by the mouse).*
Procedure **FlyingCells** (Var ACol, ARow: Integer);

*//** Return the background cell color).*
Function **GGetCellColor**(Var ACol, ARow: Integer) : Integer;

*//** Return the width of a column .*
Function **GGetColWidth**(Index: Integer) : Integer;

*//** return the column and the row number of the current cell*
Procedure **GGetCurrent** (**Var** ACol, ARow: Integer);

*//** Return of the cell textcolor*
Function **GGetTextColor** (Colum, Row: Integer) : Integer ;

*(** Return the height of the row).*
Function **GGetRowHeight**(Index: Integer) : Integer;

*//** Return the column which changed*
Function **GGetColChanged** : Integer;

*//** Return the row which changed*
Function **GGetRowChanged** : Integer;

*// ** Return the number of rows in the grid*
Function **GGetRowCount** : Integer;

*// ** Return the number of column in the grid*
Function **GGetCowCount** : Integer;

Functionalities oriented Read/Write

*//** Delete the grid content*
Procedure **GClearGrid**;

*//** Retrieve the text string of a cell*
Function **GGetText**(ACol, ARow: Integer) : string;

*//** Write a text in a cell.*
Procedure **GSetText** (ACol, ARow: Integer; Value : string);

*//** Write a text in the current cell*
Procedure **GSetTextCurrent** (Value : string);

*//** Return the number of columns in the grid.*
Function **GGetColCount** : Integer;

*//** Return the number of Row in the grid.*
Function **GGetRowCount** : Integer;

// Enable to write Unicode characters in the grid
Procedure **GsetUnicode** (Value : Boolean) ;

// Enable/disable entry in the grid cell by double clicking.
Procedure **SetDoDBclick** (Value : Boolean) ;

Multi-Media

Visual I/O allows executing multi-media instructions very simply.
We can use these functions to play sounds (* Wav or * Mid) and video animations (* AVI).

To play a media file, it is necessary to define a framework where the media will be displayed (for example, AVI). Every right-angled shape can be used (Rectangle, BMP, Support etc.). This framework will inform 'Media-Player' of its display delimitation.

The initialization of the player (InitPlayer) is in general required with the argument Streched=True.

The 3 principal instructions are:

InitPlayer : Initialization of the multi-media zone.
PlayMedia : Play a media.
NewMedia : Indicate a new multi-media file to play.

The other instructions are additional orders (stop, pause, resume, rewind etc...)

Simple program to play a file of AVI.

In a rectangle:

```
InitPlayer(' ',True);
NewMedia('trumpet.Avi');
PlayMedia;
```

Instructions List

*//** Steps backward in the currently loaded medium.*
Procedure **BackMedia**;

// Specify the length of the medium in the open multimedia device.
Function **GetLengthMedia** : LongInt;

// Return the current position within the currently loaded medium
Function **GetPositionMedia** : LongInt;

// Initialize a rectangular zone used as a framework for the MediaPlayer.
Procedure **InitPlayer** (Name : String; Stretch : Boolean);

// Indicate whether the current media is entirely played}
Function **MediaAtEnd** : Boolean;

// Indicate whether the Media in progress has been played and is at the beginning
Function **MediaAtZero** : Boolean;

// Indicate whether the current Media is playing
Function **MediaPlaying** : Boolean;

// Assign a new file name at the multi-media to played.
Procedure **NewMedia** (Nom : String);

// Skip to the next track
Procedure **NextTrack**;

// Pause playing or recording. If media has already been paused when clicked, resumes playing or recording.
Procedure **PauseMedia**;

// Pause the open multimedia device. If the device is already paused when PauseOnly is called, the device will remain paused.
Procedure **PauseOnlyMedia**;

// Play the media loaded in the open multimedia device.
Procedure **PlayMedia**;

// Set the current position to the beginning of the previous track
Procedure **PreviousTrack**;

// Resume playing or recording the currently paused multimedia device
Procedure **ResumeMedia**;

// Set the current position to the beginning of the medium,
Procedure **RewindMedia**;

// Move forward a number of frames in the currently loaded medium.
Procedure **StepMedia**;

// Stop playing or recording
Procedure **StopMedia**;

// Play a .WAV ou .Mid File
Procedure **PlaySound** (WavFileName : String) ;

// Generate a message beep.
Procedure **Beep** ;

DDE management

In addition to its DDE components, Visual I/O allows to control a DDE link by program. Attention only one channel DDE can be open in a program.

// Open a DDE channel
Procedure **OpenDDE** (Server : String) ;

// Close DDE Channel
Procedure **CloseDDE** ;

// Retrieve a data from the DDE server open with OpenDDE
Function **DDEGetData** (Topic, Item : String) : String ;

// Send a data into the DDE server open with OpenDDE
Function **DDESendData** (Topic, Item, Value : String) : String ;

// Send an Macro-command to the DDE server open with OpenDDE function
Function **DDESendMacro** (Macro : String) : String ;

☞ **Note** See also the DDE Component.

```
Var
  TS : String ;
  Data : Array [1..10] Of String ;

If Ok then
Begin
   OpenDDE('C:\excel\Excel.exe') ; {** Open the Excel server **}
   For I :=1 to 10 Do          {** Retrieve the 10 cells of column 1 **}
   Begin
      TS :='L'+IntToStr(I)+'C1' ;         {** Constitute the target by program **}
      Data[I] :=DDEGetData('Sheet1',TS) ; {** Get the target **}
   End ;
  CloseDDE ;                  {** Close the DDE channel **}
End ;
```

Serial comport management

Thanks to Serial of Visual I/O, VPU Unit allows managing the serial connections COM1 to COM10.
☞ **Note** See the components Serial Com Receiver and serial coms transmitter in the serial tab of the toolbox.

Instructions list of Serial.Vpu

// Open a serial comport
Function OpenCom (ComPort,Baudrate,Parity,Bits,Stop:String): Boolean;

// Indicate whether the specified serial comport is already open
Function GetComOpen (Comport : Integer) : Boolean;

// Close the serial comport
Procedure **DisableCom** (ComPort : Byte);

// Close all serial comport
Procedure **DisableAllComs**;

// Clear the reception buffer
Procedure **ClearCom** (ComPort : Byte);

// Indicate whether a character is present in the reception buffer
Function **V24DataOk** (ComPort : Byte) : Boolean;

// Retrieve the characters number in the reception or emission buffer
Function **ComBufferLeft**(ComPort : Byte; IO:Char) : Word; // IO='I' ou 'O' I=Input O=Output}

// Retrieve a character in the reception buffer
Function **GetByte** (ComPort : Byte) : Byte;

// Send a character in the transmission buffer
Procedure **SendByte** (ComPort,Data : Byte);

// Send a series of bytes in the transmission buffer
Procedure **ComWriteBuf** (ComPort:Byte; Var PointerTable; NB : Integer);

// Send a text string in the transmission buffer
Procedure **ComWrite** (ComPort :Byte; Value : String);

// Indicate whether the buffer is ready to be sent
Function **ReadComOk**(ComPort:Byte) : Boolean;

// Set a internal flag (general purpose)
Procedure **SetComBusy** (Comport: Byte);

//Reset the internal flag (general purpose)
Procedure **ResetComBusy** (Comport: Byte);

// Set or reset the DTR signal
Procedure **Set_DTR** (ComPort: Byte; OnOff: boolean);

// Set or reset the RTS signal
Procedure **Set_RTS** (ComPort:Byte; OnOff: boolean);

// Indicate the CTS state
Procedure **Read_CTS** (ComPort: Byte; var cts_value: byte) : Boolean;

☞ **Note** : For program examples see the chapter : Instructions for the serial comport.

Printer management

Visual I/O allows printing text Strings more simply than the traditional tools of Windows.
This library gives access to simple instructions allowing filling a printing buffer on your printer.
The printing buffer initialized by the **InitPrinter** instruction, and the printing start at the end of the treatment by the **FPrint** instruction.
The printing works compared to a paper format dependent on the default printer set.

To work with another printer or in another format (Landscape for example) you can use the printing manager of Windows to set the desired context.
You can know the number of useful pixels on your printer before or during printing by reading the variables:

XPrintPage : Number of pixels in X printable (Width usable).
YPrintPage : Number of pixels in Y printable (Height Usable).

Procedure **SETPRINTERFORMAT** (Portrait: Boolean); Allows setting your current printer in portrait or landscape.

To position the head of printing at a given place use the **SetMarge** order.
This one fixes the left margin.

Instructions List

// Initialization of the internal buffer of Visual I/O
Procedure **InitPrinter**;

// Perform a form feed
Procedure **FormFeed**;

// Send the internal buffer to the printer
Procedure **FPrint**;

// Retrieve the number of printable lines in a given font
Function **GetMaxLines** (WFont : String; WSize : Integer) : Integer;

// Retrieve the text height in a font at a given size
Function **HPrintText** (Text,Font : String; Wsize : Integer) : Integer;

// Retrevie the text width in a font having a given size
Function **LPrintText** (Text,Font : String; Wsize : Integer) : Integer;

// Send a text to the printer internal buffer
Procedure **Print** (S : String);

// Set the attributes of printed characters
Procedure **PrintAttr** (A : Integer);

// Print a Bitmap at X pixel offset from the left margin
Procedure **PrintBitMap**(XPos : Integer; BitmapName : String);

// Print a circle with a given radius
Procedure **PrintCircle** (XC,YC,R : Integer; Epaisseur : Integer);

// Change the current printing color
Procedure **PrintColor** (Color : Integer);

// Change the printing font
Procedure **PrintFont** (N : String);

// Print a line
Procedure **PrintLine** (Xo,Yo,Xe,Ye : Integer; Width : Integer);

// Send a text followed by Cr and Lf in the Printer internal buffer
Procedure **PrintLn** (S : String);

// Set the printed characters size
Procedure **PrintSize** (S : Integer);

// Set the left margin
Procedure **SetMarge**(X : Integer);

// Set the position of the pencil in the current sheet to print
Procedure **SetPen** (Y : Integer);

// Indicate the value of width (in pixels) of the currently printing page
Variable **XPrintPage** : Integer;

// indicate the height (in pixels) of the currently printing page.
Variable **YPrintPage** : Integer;

PrintAttr (Printing attributes settings)

Action Sets the characters attributes.

Syntaxe **Procedure** PrintAttr (**A** : Integer);

Description
This procedure allows setting the attributes of the characters printed.
These procedures can be called several times in a program to change the attributes of the next printed characters. By default, this attribute is **Normal**.

A Attributes of the printed characters.

Can be one of the following values:

Normal	= 1;	Regular	= 2;
Bold	= 3;	Italic	= 4;
Underline	= 5;	StrikeOut	= 6;

Limites InitPrinter must be done.
Example PrintAttr(Bold);

SetPrinterFormat (Portrait : boolean) ; *// ** If Portrait is true then printing in portrait else in Landscape*
Ex : If Start_Visu Then **SetPrinterFormat(**false) *; // *** Printing in Landscape*

Open a file the print its contents

```
Var
 LMAX,C : Integer;
Begin
 FiletextOpen(Fic,'NOTE.TXT');
 InitPrinter;
 LMax:=GetMaxLines('ARIAL',10);
 PrintSize(10);
 Println('NOTE.TXT');
 C:=0;
 While Not(Eof(Fic)) Do
 Begin
   Readln(FIC,TS);
   Println(TS);
   C:=C+1;
   If C>=LMax Then
   Begin
     FPrint;
     C:=0;
   End;
 End;
 FPrint;
 FileTextClose(Fic);
End;
```

Instructions concerning trends

A trend viewer can be used as if it was with the different components in the toolbox. However, if you need to manage a custom display, you can use directly the instructions managing the curves. For using a new curves viewer, you must use the **CustomTrend** procedure showing later. This instruction is used behind a window control.

The viewer can include trends up to10 with different colours.
The curves are displayed regardless the Y scale.

A zoom on Y can be performed by changing the Y-axis properties with the procedures **TSetLRangeY** and **TSetHRangeY**.

A zoom in X can be made only by program because each point is recorded by programmatically on axis X. Axis X does not have a scale which can contain times or numerical values (Format Abacus).

Several lines can be assigned to the texts string in the X graduations. You can insert a < CR>+<LF> into the text to be displayed.

A **High limit line** can be showed permanently by the procedure **TSetLLevel** and **TSetHLevel.**

A vertical Left and Right bar can be show permanently by the procedure **TSetBLeft** and **TSetBRight**

A function returns the equivalent value under the mouse cursor when this one is on the curve.
A function returns the point number under the mouse cursor.
A function also indicates whether the mouse pointer is moving on the curve, thus allowing starting a specific calculation.

A curve can have points of various colors, which is enough to change color before executing the **TSetPoint** procedure
From this point, the point is recorded in the curve with its position, value and color.

Instructions

Functionality: Initialization

Function **CustomTrend** (NB_Col,NB_Rang,NB_ColFixe,NB_RowFixe : Integer) : Integer;
(Assign a trend viewer in a window control and initialize it).

Procedure **TSetPencilColor** (N : Integer, Color : Integer);
(Create a pen for drawing a trend).

Procedure **TSetLRangeY** (V : real);
(Change low value of the Y axis).

Procedure **TSetHRangeY** (V : real);
(Change high value of the Y axis).

Procedure **TSetLLevel** (V : real);
(Change low line level on the Y axis).

Procedure **TSetHLevel** (V : real);
(Change high line level on the Y axis).

Procedure **TSetLLevelCol**(Color : Integer);
(Change low line level on the Y axis).

Procedure **TSetHLevelCol** (Color : Integer);
(*Change* color of the high line level on the axis Y).
Procedure **TSetDecimals** (Decimals : Integer);
(Change the number of decimals on the Y axis)

Procedure **TSetBorderRight** (Border : Integer); *//Change the number of pixels of the last graduation on the X axis*

More easy
Use new components
in trend tabs

Procedure **TSetBLeftCol**; *// Set the color of the left vertical bar)*
Procedure **TSetDIVX** (NDivX : Integer) ; *// *** Change the number of division in the main X axis ****
Procedure **TSetDIVY** (NDivY : Integer) ; *// *** Change the number of division in the main Y axis ****
Procedure **TSetBRightCol**; *//Set the color of the right vertical bar)*

Functionality: navigation

Procedure **TSetPoint** (Pen : Integer; X : integer; value : real);
(Display a point in X with a value)
Procedure **TShiftLeft**;
(Shift left the trend)
Procedure **TShiftRight**;
(Shift right the trend)
Procedure **TSetBLeft**;
(Set the position of the left vertical bar)
Procedure **TSetBRight**;
(Set the position of the right vertical bar)
Function **TGetSelection** : Integer ;
(Return the selection made by the mouse)

Functionality: Control
Procedure **TDisplayGrid** (D : Boolean);
(Show or hide the grid)
Procedure **TEnableDisplay** (On_Off : Boolean);
(Allow drawing points in the grid or not).
Procedure **TSetTextX** (N : Integer, S: String);
(Set a text under a principal graduation in X).
Procedure **TSetCoef** (Pen : Integer; Coef : Real);
(Set a multiplying coefficient for a given pen).
Procedure **TResetPoints** (Stylo : byte);
(Erase all points of a trend).
Procedure **TResetAllPoints**;
(Erase all points of all trends)
Procedure **TRegisterProc** (P : Pointer);
(Record address of a procedure to modify the type of drawing)
Function **TGetHandleBmp** : HBitmap;
(Return handleof bitmap on viewer)
Procedure **TrendToReport** (SynoReport : String; ControlName : String; TrendHandle : Integer);
*// *** Affect a trend viewer to create programmatically to a report **

Functionality: Informations
Function **TGetInterX**;
(Return the number of pixels between 2 principal graduations in X)
Function **TGetNptX**;
(Return the total number of pixels in X-coordinate)
Function **TGetNptY**;
(Return the total number of pixels in Y-coordinate)
Function **TGetGraduX** (N : integer) : String;
(Return the text associated with a principal graduation in X)
Function **TYMouse** : Real;
(Return the value of the mouse in Y)
Function **TPtXMouse** : LongInt;
(Return the value of X point pointed by the mouse)
Function **TMouseMove** : Boolean;
(Return True if the mouse moved on the trend)
Function **TGetDivX** : Integer ;
(Return the number of principal divisions in X }
Function **TGetDivY** : Integer;
(Return the number of principal divisions in Y }
Function **TGetYMax** : Integer;
(Return the max value of the Y axis }
Function **TGetYMin** : Integer; *// Return the Min Value of the Y Axis*

Examples by program

```
// **** In global variables ****
Uses Windows,Sysplc ;
Var
  Point     : Integer;
  Grille    : Boolean;
  Valeur    : Real;
// **** In Window 1 ***************
If Start_Visu Then
Begin
  Customtrend(5,3,2,1,0,300); // Viewer declaration
  TsetPencilColor(1,0);      // colour black for pen #1}
  Point:=0;
End;
// *** Show/hide the grid by clicking with the mouse
If ToplBdown Then             // Left click with the mouse
Begin
  TDisplayGrid(Grille);       // Display the Grid
  Grille:=Not(Grille);        // flip-flop on the grid
End;
// * Displaying of the value (random) every 100 ms *
If BT_100=1 Then              // drawing every 100 ms}
Begin
  Valeur:=Random;              // calculation of Y with random value
  TSetPoint(1,Point,valeur);  // Drawing of the point
  Inc(Point);                 // Increment of the abscise}
  If Point>TGetNptx Then      // Arrive at the right end of the viewer
  Begin
   TshiftLeft;          // Shifting left, sliding curve
   Point :=Point -TgetInterX;      // recomputing the point
  End;
End;
```

TrendToReport

```
// *** Window1 ************************************
SObject Procedure Window1;
Var
 THDLE : Integer;
Begin
  If Start_Visu Then
  Begin
    THDLE:=CustomTrend(10,5,1,1,0,450); // ** Get the viewer handle
    TSetPencilColor(1,255);
    TSetPoint(1,10,10);
    TSetPoint(1,100,100);
    TSetPencilColor(2,1255);
    TSetPoint(2,20,20);
    TSetPoint(2,200,200);
    TrendToReport('report','Window1',THDLE); //Form Rapport.syn, Window1 control
  End;
End;

// *** Button1 ************************************
SObject Procedure Button1;
Begin
  If Button Then PrintReport('Report');
End;
```

Customisation of points in the trends

Components seen previously call the Visual I/O instructions for the graphic management of curves.
The principle adopted for the customisation of drawings is :

- If any **CallBack** address was specified, straight lines draw curves between every points fixed by program (**TSetPoint**).

- If a CallBack address is specified, the graphic is not any more made by the system. But the CallBack procedure is called. The system passes to it a structure containing information about the point that must be drawn.

The StrucPaint Structure

This structure will be used with the graphic instructions of the Windows API (GDI.DLL).
Refer to Microsoft documentations SDK.

```
//******** Custom trend ******
  StrucPaint = Record
       PDC : Integer;  // ***** Drawing Context **
       X   : Integer;
       Y   : Integer;
       Color      : Integer;
       OrigineX   : Integer;
       OrigineY   : Integer;
       XBefore    : Integer;
       YBefore    : Integer;
       Pen        : Integer;     // # Pen ********
       Value      : Extended;
       ValueBefore  : Extended;
       Canvas       : Integer; // Delphi TCanvas Compatible
       FirstPoint   : Boolean;
       LastPoint    : Boolean;
  End;
```

More easy
Use new components
in trend tabs

```
PDC          : Display Context
X ,Y         : Graphical coordinates of the point to display.
Color        : Color of the Point to display.
OrigineX,
OrigineY     : Intersection coordinates of the Axis X & Y of the trend.
Xbefore,
YBefore      : Graphical coordinates of the previous point displayed.
Pen          : Pen number ( 1 up to 10 ).
Value        : Numerical value of the point.
Canvas       : Display Canvas (Borland Delphi compatibility).
Firspoint    : True if it is the first point displayed else false.
LastPoint    : True if it is the last point displayed else false.
```

Graphics.VPU

This library contains some procedures and graphic functions based on Windows API. It also contains some examples of CallBack procedures.

To call a custom procedure (CallBack) you can use the procedure TRegisterProc and pass the address of your own drawing procedure in parameter.

Example :

```
Var
  PT : Integer;
  PT:=Addr(PtToCircle);  // Get the PtToCircle address
  TRegisterProc(PT^);   //  Affect the callback to the PtToCircle procedure
```

Addr(PtToCircle) Do not return directly the address of the procedure but return directly the memory address containing the final address of the procedure (indirection or pointer)
PT^ means the content of the memory addressed by the value of PT.
Example of drawing procedure in Call back (Extract of Graphics.Pas).

```
// ******* Custom Trend Pts to Circle ******
Procedure PtToCircle;
Begin
 Ellipse(StrucPaint(PAINTSTRUCT).PDC,StrucPaint(PAINTSTRUCT).X-2,
         StrucPaint(PAINTSTRUCT).Y-2,
         StrucPaint(PAINTSTRUCT).X+2,
         StrucPaint(PAINTSTRUCT).Y+2);

 Line( StrucPaint(PAINTSTRUCT).PDC,
       StrucPaint(PAINTSTRUCT).XBefore,
       StrucPaint(PAINTSTRUCT).YBefore,
       StrucPaint(PAINTSTRUCT).X,
       StrucPaint(PAINTSTRUCT).Y);
End;
```

☞ **Note** StrucPaint(PAINTSTRUCT).PDC `// *** Typecasting of PainStruct **`

TGetSelection

Action Return a pointer on a structure of TRECT type.
Syntax TGetSelection : **Integer**;
Return Address of TRECT structure of the current selection.
Description
This function returns the address of the structure containing information of the last selection to the mouse by the operator. The operator can select a rectangle (in rectangle) defining a zone in the viewer of curves

The Trect type is defined in Graphics.pas like below:

```
TRECT = Record
        Left   : Integer ;
        Top    : Integer ;
        Right  : Integer ;
        Bottom : Integer ;
    End ;
```

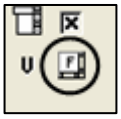
More easy
Use new components
in trend tabs

Example

```
Uses Graphics ;
Var
  PT :Integer ;
  TR : TREct ;
Begin
   PT:=TGetSelection;
   Move(PT^,TR,Sizeof(TR));
End;
```

Web navigator

The Web navigator is obtained by transformation of a window control from a form into a window allowing viewing HTML or PDF pages.



```
// ** Create a Web navigator in a window control
Function WEBBROWSERCREATE : Integer;

// ** To display a HTML or PDF file.
Procedure WEBNAVIGATE (Browser : Integer; URL : String);

// *** Go to the previous page
Procedure WEBGOBACK ( Browser : Integer);

// *** Go to the next page
Procedure WEBGOFORWARD ( Browser : Integer);

// *** Go to the home page
Procedure WEBGOHOME ( Browser : Integer);

// *** display the page of searching on the web
Procedure WEBGOSEARCH ( Browser : Integer);

// ** To refresh the current page
Procedure WEBREFRESH ( Browser : Integer);

// ** To Stop displaying the current page
Procedure WEBSTOP ( Browser : Integer);
```

Program example

```
SObject Procedure Window5;
Var
 MyBrowser : Integer; // *** Browser Handle
Begin
  // **** Web Browser Object **************
  If Start_Visu Then
  Begin
    MyBrowser:=WEbBrowserCreate;
  End;
   // **** Menu Load File **********
  If LFile Then
  Begin
    LFile:=False;
    HURL:=Open_File(HFileName,'Html Files','*.html','Choose an HTML File');
  End;
  // **** URL to Navigate *************
  IF HURL then
  Begin
    WebNavigate(MyBrowser,HFileName);
  End;
  // ***** Stop Display HTML Page *******
  If HStop Then
  Begin
    HStop:=False;
    WebStop(MyBrowser);
  End;
  // **** Go Next Page ***********
  If HFORWARD Then
  Begin
    HFORWARD:=FAlse;
    WEBGOFORWARD(MyBrowser);
  End;
End;
```

☞ **Note**: This example is available in the visual I/O CD-ROM after installation.

Date and time management

Visual I/O contains multiple procedures and function for the management of the dates and the times.

Generalities

Visual I/0 uses the Real type (extended type) for the calculation of dates and times.

The Variables CDate and CTime are available to the programmer in reading.
These global variables are recurrently refreshed all 500ms.
These values are also available in String Type: SCDate and SCTime.

CDate	: Real
CTime	: Real;
SCDate	: String]; {*** Ex: 12/04/1994 ****}
SCTime	: String;{*** Ex: 15:45:34 ******}

With these four variables, it is possible to display the date, the time and also to calculate other dates or times.
The format of ScDate depend on sShortDate in the section [intl] from file WIN.INI.

The calculation of the date and the time is as follows:

Hour:= ((Hour X 60 + Min)X 60000) + Sec X 1000 + MSec) / MSecsPerDay;

MSecsPerDay = 24 * 60 * 60 * 1000;

The base of calculation is the millisecond.

For example, I want to know the current date minus 4 Hours:

```
Var
  TH     : Real;
  DateP : String;

TH:=((4 *60)*60000)/MSecsPerDay;    {** Calculate the number of milliseconds **}
Previous:=(CDate+CTime)-TH;   {** Subtraction to the current date and time **}
DateP:=DateTimeToStr(Previous);     {** Convert in String ***}
```

☞ **Note**: To subtract days you must convert them into hours.

Instructions for Date and Time

{Convert a DateTime value to a string
Function **DateTimeToStr** (DateTime: Real) : String;

{ Return the day of week }
Function **DayOfWeek** (Date: Real): Integer;

{ Return a DateTime value that represents a specified Year, Month, and Day }
Function **EncodeDate** (Year, Month, Day: Word) : Real;

{ Return a TDateTime value for a specified Hour, Min, Sec, and MSec }
Function **EncodeTime** (Hour, Min, Sec, MSec: Word) : Real;

{ Return the current date and time }
Function **Now** : Real;

{ Convert a string to a TDateTime value }
Function **StrToDateTime** (S: string) : Real;

{ Convert a string to a DateTime value. }
Function **StrToTime** (S: string) : Real;

{ Return a string that represents a DateTime value }
Function **TimeToStr**(Time: Real) : String;

{ Convert a Date into String }
Function **DateToStr**(Date: Real) : String;

// Format a TDateTime value
Function **FormatDateTime** (Format : String; DateTime : Real) : String ;

Specification Display

c Display the date, by using the format defined by the global variable ShortDateFormat, then the time by using the format defined by the global variable LongTimeFormat.
The time is not displayed if the fractional part of DateTime is equal to zero.
d Display the number of the day without a zero in front (1 to 31).
dd Display the number of the day with a zero in front (01 to 31).
ddd Display the day shortened (Sun to Sat) by using the Strings provided by the global variable ShortDayNames.
dddd Display the complete day (Sunday to Saturday) by using the Strings provided by the global variable LongDayNames.
ddddd Display the date by using the format provided by the global variable ShortDateFormat.
dddddd Display the date by using the format provided by the global variable LongDateFormat.
m Display the number of month without a zero in front (1 to12). If the specification m immediately follows a specification h or hh, the minutes are displayed rather than the number of the month.
mm Display the number of month with a zero in front (01 to 12). If the specification m immediately follows a specification h or hh, the minutes are displayed rather than the number of the month.
mmm Display the month shortened (Jan to Dec) by using the Strings provided by the global variable ShortMonthNames.
Mmmm Display the complete month (January to December) by using the Strings provided by the global variable LongMonthNames.
yy Display year with two digits (00 to 99).
yyyy Display year with four digits (0000-9999).
h Display hour without a zero in front (0 to 23).
hh Display hour with a zero in front (00 to 23).
n Display minutes without a zero in front (0 to 59).
nn Display minutes with a zero in front (00 to 59).
s Display secondes without a zero in front (0 to 59).
ss Display minutes with a zero in front (00 to 59).
t Display time by using the format provided by the global variable ShortTimeFormat.
tt Display time by using the format provided by the global variable LongTimeFormat.
am/pm Use the format over 12 hours with the specification h or hh which precedes it.
Display 'am' for the hours before midday and 'pm' for the hours after midday.
The specification am/pm can indifferently use the capital letters or the small letters.
a/p Use the format over 12 hours with the specification h or hh which precedes it.
Display 'a' for the hours before midday and 'p' for the hours after midday.
The specification a/p can indifferently use the capital letters or the small letters.
ampm Use the format over 12 hours with the specification h or hh which precedes it.
Display the contents of the global variable TimeAMString for the hours before midday and the contents of the global variable TimePMString for the hours after midday.
/ Display the character of separation of date provided by the global variable DateSeparator.
: Display the character of separation of time provided by the global variable TimeSeparator.
'xx'/"xx" The characters placed between ' ' or " " are displayed just as they are and do not affect the format.

The specifications of format can be indifferently in capital letters or in small letters, the result obtained is identical.
If the string provided by the parameter Format is empty, the date and the time are implicitly formatted with the specification of format ' C '.

<u>Example :</u>
The following example affects the string ' Go Friday, February 13, 1997 to 10:30 AM' to the variable S.

```
S := FormatDateTime('"Go " dddd, mmmm d, yyyy, ' +  '"to" hh:mm AM/PM', StrToDateTime('13/2/97
10:30am'));
```

Text file management

Simply put *text files* containing readable ASCII characters. We can think of working with text file in Visual I/O as analogous to playing or recording information on a cassette tape.
Although it is possible to make changes within text file, jump around when processing information or add some data to the file other than at the end, it is advisable to use a text file only when we know that we are working with ordinary text and no such operations are necessary.
Text files are considered to represent a sequence of characters formatted into lines, where an end-of-line marker (a carriage-return / linefeed combination) terminates each line. Just think of Win.ini or Autoexec.bat.
For example, to get the ligne #3 in text file, you need to readothe line #1 then the line #2 then the line #3.

instructions List

// Concatenation of 2 files
Function **FileConcat** (Source, Destination: String) : Boolean;

// Copy a file to another file
Function **FileCopy** (Source, Destination : String) : Boolean;

// Clear a file
Function **FileDelete** (FileName: String) : Boolean; *// ** Depuis Version 7 est dans Gestfile.VPU*

// Test whether a specified file exists
Function **FileExists** (FileName : String) : Boolean;

// Change a file name
Function **FileRename** (OldName, NewName: String): Boolean;

// Prepare a file for adding text to the end if the file do not exists the system creates it
Procedure **FileTextAppend** (Var F: Integer; Name : String);

// Close a specified file.
Procedure **FileTextClose** (Var F: Integer);

// Open a specified file using a reading access mode
Procedure **FileTextOpen** (Var F: Integer; Name : String);

// The Readln procedure reads a line of text and then skips to the next line of the file.
Procedure **ReadLn** (Var F : Integer; St : String);

// Write writes one or more values to a text file.
Procedure **Write** (Var F : Integer; St : String);

// After executing Write, Writeln writes an end-of-line marker (carriage return/line feed) to the file
Procedure **WriteLn** (Var F : Integer; St : String);

// Test whether the file position is at the end of a file
Function **EOF** (Var F : Integer) : Boolean ;

Example of reading a text file

```
Var
  Fic : Integer ;
  TS  : String ;
Begin
  If FileExists('Readme.txt') Then
  Begin
    FileTextOpen(Fic,'Readme.txt');
    While Not(Eof(Fic)) Do
    Begin
      Readln(Fic,TS);
      Addstring(TS);
    End ;
    FileTextClose(Fic);
  End;
End;
```

Access to global variables by functions

Function **GetVpuValue**(NomVar : String) : String;
*// ** Return the value of the requested variable in the form of a textr string.*
Procedure **SetVpuValue**(Varname, Value : String) ;
*// ** Modify the value of the variable named in VarName.*

```
Example:                SetVpuValue( 'W0','123') ;
                        SetVpuValue( 'B0,'TRUE') ;
```

Function **GetVpuAddr**(NomVar : String) : Integer;
*// ** Return the memory address of the specify variable.*
The variables are lined in memory consecutively (Packed) by recovering the address of the variable and knowing the declarations of the other variables. It is possible to handle all the variables by block instead of not one by one.
Function **GetProjectDir** : String; *// ** return the path of the current project*

Others

Procedure **VPLOAD** (VPUFileName : ShortString);
*// *** Load a program or a library compiled by Visual I/O or Visual PLC ****
Example :
VPLOAD (TASK1.VPU'); *// *** .VPU Extension is necessary*

Function **VPUNLOAD** (VPUFileName : ShortString) : Integer;
*// *** Unload a program or a library from the memory ***
VPUNLOAD ('TACHE1'); *// *** Without extension*

Function **ExtractRes** (**FileNameinEXE :** String; **DestinationFile** : String) : **Boolean**;
*// ** Extract a file from the executable made by Visual I/O*

FileNameinEXE	File name stored in the executable file.
DestinationFile	Name and path of the file after extraction.
Return	Extraction done.

Example :
```
Ok:=ExtractRes ('Menu.HTM','C:\Doc\Menu.htm') ;
```
'Menu.HTM' is the name of the resource (file) in the current executable.
The executable named is the program containing the instruction. It is not an external program.

☞ **Note** : You can show the resources incorporated in the executable with the utility named SCANRESEXE.EXE

Procedure **WinDialog** (Value : Boolean);
*// *** Transform the current form in Dialog mode, on the other hand, the programs in the form will be executed only if the form has the focus (title bar active).*

Send an Email by program

Function **XSendEmail** (MailFrom, MailRecipient, MailSubject, MailBody, SmtpHost : String) : String;

MailFrom	: Email provenance.
MailRecipient	: Destination Email.
MailSubject	: Subject of the email
MailBody	: Title of the email
SmtpHost	: SMTP Server used to send the email. (see outlook express configuration)

Return value : Empty string if ok, else returns the text of the error.

Example :
XSendEmail('Visual I/O Application ', 'holine@arsoft.eu', 'congratulations', Send an email is very easy', 'smtp.hotmail.com');

pop up menus

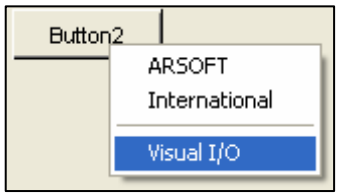
These procedures and functions enable creating Popup menus dynamically (showed by the right click of the mouse). This functionality is interesting when it is necessary to change text according to the context or the component on which the mouse is. For the test we use the same component and instructions as the main menu.

Function **POPUPMenuCreate** : Integer;
*// ** Create an empty pop up menu*

Procedure **POPUPMenuDestroy** (Handle : Integer);
*// ** Destroy a popup menu*

Procedure **POPUPMenuAddChild** (Handle : Integer; Text : String)' *// **** Text = Text + , +Identifier;*
*// ** Add items to the popup menu*

```
// *** Button1 ************************************
SObject Procedure Button2;
Var
 HP :Integer;
Begin
  if Button Then
  Begin
    HP:=POPUPMenuCreate;
    POPUPMenuAddChild(HP,'ARSOFT,300');
    POPUPMenuAddChild(HP,'International,301');
    POPUPMenuAddChild(HP,'-,0');
    POPUPMenuAddChild(HP,'Visual I/O,302');
  End;
End;
```



Load and save an ActiveX context.

*// ** to know the ActiveX name behind this function which is executed.*
Fonction **OCXGetClassName** : String;

*//*** Save the current properties of the ActiveX to reload them later*
*//*** Current Graphical properties (colours, fonts, size etc..)*
Procedure **SavePropOcx** (PropName : String);

*// *** Load the properties in the current Activex. The properties have been saved before with the SavePropOcx procedure.*
Procedure **LoadPropOcx** (Nom : String);

Changing the text font and its style

*// ** Change the font name of the component*
Procedure **SetFontName** (FName : String);

*// ** Change the font style of the component*
Procedure **SetFontStyle** (FStyle: Integer);

Fstyles : Bold = 0
Italic = 1
Underline = 2
StrikeOut = 3

Data base management with BDE

Visual I/O integrates the BDE (Borland DataBase Engine) to sort, add, erase... data or recordings in tables of the type PARADOX, DBASE or others according to the type of ODBC driver installed and chosen.
The creation and the modification of tables can be made with DataBase-DeskTop utility (or by Sql request).

For a very fast handling of DBASE 3+ Tables, you can use the **Microbase.VPU** library.
This library allows only simple navigation, insertion, clearing, additions and sorting on a DBASE 3+ table.
It has the advantage to be extremely fast because it manages DBASE 3+ table like a direct access file.
In certain cases Microbase suffuses. DBASE 3+ is compatible with all the office softwares.

Tables and structures of data

The creation of table allows defining the structure of the table, organisation and the type of the data. To create, modify or visualize tables, you can use Borland's DataBaseDeskTop or the example in the Visual I/O cd-rom (project create table).

Index or Keys

The system of indexation is the core of the DBMS (Data Base Management System). Without the possibilities that offers to perform fast localizations, to provide sorting instructions and to operate on connections with tables, the data contained in them would be only a list of dead bytes and not exploitable.

The Key and Index are synonymous:
The primary key of table (PARADOX) determines the sequential order of data as well as the possibility to establish links with other tables.
A simple primary key is defined in the first field of the table.
A composite primary key is defined in N first fields of the table.

For a simple key or composite key, this key must be single.

For a simple key, the same field of two different recordings cannot have the same value.
For example, a table having a simple key on the field "Name" and a field "First name" not forming part of this key will not be able to accept the individuals 'Jean Dubois' and 'Jacques Dubois', because "Dubois" can be the key of only one recording.
For a composite key, all of the values contained in the key must be overall single.
With the same example as previously, but with a primary key on "Name" and "First name", the table will be able to contain at the same time 'Jean Dubois' and 'Jacques Dubois', or 'Jean Dubois' and 'Jean Durant', but 'Jacques Dubois' not twice.

Programming

A Data base is generally a directory containing tables with formats such as PARADOX, DBASE, Access, etc.
Visual I/O allows navigating, modifying, inserting, writing records in these tables.
This navigation is possible through an object created by program. This object enables some methods (Functions and Procedures).
The object above must be created and indicated on the database that must serve and on the table on which it must work. The functions of this object are contained in BDE.VPU.

Example of the declaration of object interface for database:

In the global variables of your form, declare:

```
Uses BDE ; { The functions are in this VPU **}
Var
  MyBase : Integer;  {*** Mybase is the table Handle ****}

//** In an unspecified component or program:
If Start_Visu Then
Begin
   Mybase:= DBCreate ('C:\Base\Cumul.DBF');{* Data Base in the directory C:\Base
End;
If End_Visu Then {*** At the end of the program, closing of the base ****}
      DBClose(MyBase);
```

Note:
While the database is not closed (DBClose), the new introduced data are not truly written on the disc.
Attention: several channels can be open at the same time.

Instructions List

Function **DBOk** (Base : Integer) : Boolean; *{ Test whether the base and the table in progress are correctly open.}*
Procedure **DBClose**(Base : Integer); *{ Close the base and the table in progress.}*
Procedure **DBCreate** (Table: String); *{ Create an interface with a data base and a table.}*
Function **DBAppend** (Base : Integer): Boolean; *{ Add a new recording at the end of the table to progress.}*
Function **DBBof** (Base : Integer): Boolean; *{ Indicate whether the table is on the first recording.}*
{ Validate the confirmation message for erasing a record or not.}
Function **DBConfirmDel** (Base : Integer, Confirmation : Boolean);
Function **DBDelete** (Base : Integer): Boolean; *{ Delete the current record.}*
Function **DBDeleteTable** (Base : Integer): Boolean; *{ Delete the current table.}*
Function **DBEmptyTable** (Base : Integer): Boolean; *{ Erase all the records of the current table.}*
Function **DBEof** (Base : Integer): Boolean; *{ Indicate if the table cursor is on the last record.}*
{ Execute a SQL or QBE request stored in a file.}
Function **DbexecQuery** (Base : Integer ; Name : String) : Boolean;
Function **DBFind** (Base : Integer ; S: String):Boolean; *{ Seek the first field containing the element requested.}*
{ Seek the first field approaching the element requested.}
Function **DBFindNearest** (Base : Integer ;S : String) : Boolean;
Function **DBFirst**(Base : Integer) : Boolean; *{ Position the current table on the first record.}*
Function **DBGetCount**(Base : Integer) : Integer; *{ Return the number of records in the table.}*
{ Return the position of the current record in the current table.}
Function **DBGetCurrentPosition** (Base : Integer): Integer;
{ Return the contents of the field specified in the form of string in the current record.}
Function **DBGetField** (Base : Integer ; FieldNumber:Integer) : String;
{ Return the contents of field in the current table.}
Function **DBGetFieldByName** (Base : Integer ; N:String) : String;
{ Return the last error occuring in the BDE.}
Function **DBGetLastError** : String; *{** Common to all the databases opened **}*
{ Return a field name in the current table.}
Function **DBGetNameField** (Base : Integer ; N:Integer) : String;
{ Return the index name in the current table.}
Function **DBGetNameIndex** (Base : Integer ; N:Integer) : String;
{ Return the number of fields in the current table.}
Function **DBGetNFields** (Base : Integer) : Integer;
{ Return the number of index of the current table.}
Function **DBGetNIndexs** (Base : Integer) : Integer;
Function **DBGotoBookMark** (Base : Integer): Boolean; *{ Go to the last bookmark.}*
{Insert a record in front of the current position of the table.}
Function **DBInsert** (Base : Integer): Boolean;
{ Return the current index in the current table.}
Function **DBGetindexFieldName** (Base : Integer): String ;
{ Change the valid indexes in the current table.}
Function **DBSetindexFieldName** (Base : Integer ; IndexesNames : String);
{ Return the sorted index in progress in the current table.}
Function **DBGETindexName** (Base : Integer): String;
{Change the index in progress in the current table.}
Procedure **DBSETindexName** (Base : Integer ; IndexName: String);
{Position the table in progress on the last record.}
Function **DBLast** (Base : Integer) : Boolean;
{ Modify the current record of the table in progress.}
Function **DBModify** (Base : Integer): Boolean;
{Move the cursor by N records in a table.}
Function **DBMoveBy** (Base : Integer ; N : Integer) : Boolean;
{ Go to the next record in the table.}
Function **DBNext** (Base : Integer) : Boolean;

Function **DBPackTable** (Base : Integer) : Boolean; *{ Optimize the space of the current table}*
Function **DBPrior** (Base : Integer): Boolean; *{ Go to the prior record in the table.}*

{Write a specified field in the current table.}
Function **DBPutField** (Base : Integer ; N : Word; S: String) : Boolean;

{ Write a specified field in the current table the field is specified by its name.}
Function **DBPutFieldByName** (Base : Integer ; Name : String; S: String) : Boolean;

Procedure **DBQAdd** (Base : Integer ; S : String); *{ Add a line to a SQL request.}*

{ Indicate whether the answer table of the request is on the first record.}
Function **DBQBof** (Base : Integer): Boolean;

{ Delete the record in progress in a SQL answer table.}
Function **DBQDelete** (Base : Integer): Boolean;

{ Indicate whether the answer table of a request is on the last record.}
Function **DBQEof** (Base : Integer): Boolean;

{ Execute a replacement of a variable in a requested file.}
Function **DBQExchange** (Base : Integer ; Name,S,R : String) : String;

{ Position the answer table in progress on the first record.}
Function **DBQFirst** (Base : Integer): Boolean;

{ Return the records number in the answer table.}
Function **DBQGetCount** (Base : Integer): LongInt;

{ Return the value of a field in the answer table}
Function **DBQGetField** (Base : Integer ; FieldNumber:Integer) : String;

{ Return a field in the answer table in progress, the field is addressed by its name.}
Function **DBQGetFieldByName** (Base : Integer ;N:String) : String;

{ Return the name of a field in the answer table}
Function **DBQGetNameField** (Base : Integer ; N:Integer) : String;

{ Return the field count of the answer table in progress.}
Function **DBQGetNFields** (Base : Integer) **:** Integer;

{ Position the answer table in progress on the last record.}
Function **DBQLast** (Base : Integer) : Boolean;

{ Modify the current record of the answer table in progress.}
Function **DBQModify** (Base : Integer): Boolean;

{ Move forward or backward of N Records in a answer table.}
Function **DBQMoveBy** (Base : Integer ;N : Integer) : Boolean;

{ Go to the next record in the answer table. }
Function **DBQNext** (Base : Integer): Boolean;

{ Go back to one record in the answer table. }
Function **DBQPrior** (Base : Integer): Boolean;

{ Write a particular field in the memory buffer of the current answer table.}
Function **DBQPutField** (Base : Integer ;N : Word; S: String) : Boolean;

{ Reset a working area in the table in progress.}
Procedure **DBResetRange**(Base : Integer);

{Set a bookmark in the current table.}
Function **DBSetBookMark** (Base : Integer): Boolean;

{ Set a new message for deleting confirmation.}
Procedure **DBSetMessageDel** (Base : Integer ;S : String);

{ Set the new key (index) in the current table.}
Procedure **DBSetNindex** (Base : Integer ;N : Byte);

{ Position the cursor (Pointer) on a particular record.}
Function **DBSetPosition** (Base : Integer ;P : LongInt) : Boolean;

{ Set a working area in the current table.}
Function **DBSetRange** (Base : Integer ; St,Se : String; Exclusive : Boolean) : Boolean;

{ Sort the current table sorting is performed on the first field}
Function **DBSortTable**(Base : Integer) : Boolean;

{ Enable or Disable the display of the error messages}
Variable **StopDBMsg** : Boolean; *{* Variable common to all the opened bases *}*

QBE & SQL Requests

Visual I/O can perform SQL & QBE requests on your tables. You can record the instructions Qbe or Sql in a text file. Visual I/O in can execute as well as your Qbe or Sql requests saved in file .QBE or .SQ

Saving in a.QBE or .SQL file

After having defined your Qbe (and tested), you must save it in a file with QBE or SQL extension to use it in Visual I/O with the **DBexecQuery** instruction.

Example of SQL requests saved:

```
SELECT DISTINCT "C:\BDE\EXAMPLES\TABLES\STOCK.DB"."Stock No", Model
FROM "C:\BDE\EXAMPLES\TABLES\STOCK.DB"
WHERE
("C:\BDE\EXAMPLES\TABLES\STOCK.DB"."Stock No" > 10)
ORDER BY "C:\BDE\EXAMPLES\TABLES\STOCK.DB"."Stock No", Model
```

<u>Example of QBE request saved:</u>

```
C:\BDE\EXAMPLES\TABLES\STOCK.DB        | Stock No      | Model  |
                                       | Check >10     | Check  |
```

Dynamical QBE & SQL Requests

After having formatted and saved your requests, you can define variables in this request, which will be affected during the life of your application.
In the proceeding example about QBE request, we have tested the field Stock No.
This test allows seeking all the Stock No higher than 10.

```
C:\BDE\EXAMPLES\TABLES\STOCK.DB        | Stock No      | Model  |
                                       | Check >10     | Check  |
```

It would be interesting to affect the comparison value (here 10) dynamically with another value.
For that, we will replace value 10 with a variable named %%.

Modification and safeguard of the QBE under a text editor:

```
C:\BDE\EXAMPLES\TABLES\STOCK.DB        | Stock No      | Model  |
                                       | Check >%%     | Check  |
```

In your program you must affect the variable %% with a value before executing the Qbe request (the same principle in SQL). You must use the **DbQExchange** instruction to modify this variable. Let us imagine that you want to affect the variable %% with value 53. The instruction will be as follows:

NEW_FILE:=**DbQExchange**(Mybase,'REQUEST.QBE','%%','53');

This line of program exchanges the named variable %% with value 53 in file REQUEST.QBE and returns the file name containing this modification.

For indication and for verification, the new file will be stored in the current directory under name **A.QBE** or **B.QBE.**

C:\BDE\EXAMPLES\TABLES\STOCK.DB	Stock No	Model
	Check >53	Check

It is possible to execute the query:

DBexecQuery(Mybase, NEW_FILE);

You can rectify several variables in a query.
Example:

C:\TABLES\STOCK.DB	Stock No	Model	Part No
	Check >%%	Check dv.	Check >###

Here we have declared two Variables %% and ###, they should be replaced by program before starting the request.

%% will be replaced by Value 53 and
will be replaced by Value 120.
Program :

NEW_FILE:= **DbQExchange**(Mybase,'REQUEST.QBE','%%','53');
NEW_FILE:= **DbQExchange**(Mybase,NEW_FILE,'###,'120');
DBexecQuery(Mybase,NEW_FILE);

First line = exchange all the variables %% contained in the file REQUST.QBE with value 53 and return the name of the file containing the modification (Request.Qbe does not change).

Second line = exchange all the variables ### contained in the file name which is returned in the variable NEW_FILE with value 120 and return the name of the file containing the modification in the NEW_FILE variable.

Third line = execution of the request described in the file name stored in variable NEW_FILE.

Note: the request source (here REQUEST.QBE) is never modified. The modifications are in the files created by the **DbQExchange** function**.**

Name of the dynamic vriables:

It is possible to imagine every name of variable, example:
$$ ****))) 777 AZE VISUALIOWINDOWS
However the **DbQExchange** procedure will be able to confuse ## and ###, or VISUAL and VISUAL_IO.
Thus, it is necessary to avoid choosing the same characters to define a dynamic variable.

Answers to QBE & SQL requests

After having executed the function, DBexecQuery Visual I/O creates a virtual answer table containing the result of the executed request. You have access to a list of instructions which are very close to the traditional list of instructions, to question the table of answer.
In the list of instructions, all the functions or procedure start by:
DBQ....

Constitution of SQL Requests programmatically

It is possible to create a dynamic Sql request by program by using the **DBQAdd** procedure. Each line will be added to the internal list.
Example:

```
//** In local Variables of a form or a program **
MaBase: DataBase;
//***** In a program of a component in a form ****
If Start_Visu Then
    MaBase:=DataBase.Create('C:\Base\Clients.Dbf');
```

```
If Button Then
Begin
   DbQadd(Mabase,'SELECT Label);
   DbQadd(Mabase, 'FROM "PROVA.DB" );
   DbQadd(Mabase, 'WHERE');
   DbQadd(Mabase, 'LIBELLE= "Visual IO "');
   if DbExecQuery( Mabase, '') Then  //* Execution of the final request
    //**** Treatment ******
End;
```

Examples

Retrieve the value of a field in table:

```
//****** In Local Variables *********
MaBase: DataBase;
//***** In a program of a component ********
If Start_Visu Then
Begin
 MaBase:=DataBase.Create('C:\Base\Clients.dbf');//Directory containing the table
End;
If Button   Then SetText(Mabase.DbGetField(Mabase,1));
If End_Visu Then DBClose(Mabase);
```

Assignment of the table fields:

```
//****** In Local Variables *********
MaBase: DataBase;
//***** In a program of a component ********
If Start_Visu Then
Begin
   MaBase:=DataBase.Create('C:\Base\Clients.dbf');
End;
If Button Then
Begin
   If DBPutField(MaBase, 1,'Dupont') Then
   If DBPutField(MaBase, 2,'Champs Elysées') Then DBModify(MaBase);
End;
```

Documentation DBCreate procedure

DBCreate Procedure

Action Create an interface with a database and a table.

Syntaxe <*Labase*>:= DBCreate(**Path** : String);

Description
This function allows initializing a Handle DataBase type and informing it of the DataBase and the table that you want to address. It is possible to initialize several Database objects, but it can be used only one time as long as the DBClose function was not applied to this object. In each **DBCreate** used, a **DBClose** corresponds.
The instruction DBCreate can be used at the start of the application and the instruction DBClose used at the end of the application. However, DBCreate and DBClose are usually used with each operation on a table to guarantee the correct saving of the data.

To have an access to a DataBase table, it is necessary:

1. To declare an object of DataBase type in Local variables.
Example: MyBase : **Integer**; *{Handle MyBase of DataBase type.}*
2. To Initialize this Object.
Example:
If start_Visu **Then** MyBase:= **DBCreate**('C:\Examples\Tables');
3. To close the interface with the DataBase and to restore them into the memory
Example: **If** End_Visu **Then** DBClose;

Limits One DBClose only corresponds to one DBCreate.

Extensions Alias:
The instruction DBCreate allows working with Alias defined in the configuration (Icon Configuration Database Engine seen at the beginning of this chapter).
The DBCreate procedure allows using one alias which has already been defined.
Example :

Mybase:= DBCreate('TestExcel\Data'); *{*** work with the alias TestExcel ***}*
*{*** work with the table Data.Xls **}*

In this case, the database addressed is the way contained in the alias TestExcel and the table is DATA without corresponding type of table since the Alias already knows this type of table.

Loading and unloading DLL manually

The BDE DLL is the illustration of the loading and manual unloading of a DLL. This mechanism should not be used because Visual I/O already uses this mechanism when the programmer used procedures and functions contained in a DLL (see in following chapter).
You can use the **LoadLibrary** function in the Windows API to load a library. You can use the **FreeLibrary** procedure to unload the library from the memory. You can use the **GetProcAddress** function to find the address of the procedure or a function in this DLL.

Example: (from BDE.PAS).

```
//** Load the DLL in memory
 HDle :=LoadLibrary(PChar('Bdext'));

//** Get the Address of the DBGETIndexName procedure in the DLL
 AD:=GetProcAddress(Hdle,PChar('DBGetIndexName'));

//* Replace the address of the procedure in Visual I/O
 Move(Ad,DBGetIndexName,4);

//* Unload the DLL
 Freelibrary(HDLE);
```

Calling procedures and functions in a DLL

Visual I/O allows calling procedures and functions located in a dynamic link library DLL. This interface is interesting in the case where you have libraries for scientific cards, or if you decide to enlarge Visual I/O functionalities written under another language (for example in Visual Pascal, C++, Delphi).

To use these procedures and functions in a DLL, it is necessary to declare the structure of the function or the procedure in the INTERFACES part of a unit (vpu). The library is used as a link between Visual I/O and the DLL.

When you import a routine from a DLL, the procedure or function is imported explicitly by its name.

Let us examine the following declaration:

```
Unit
Interface
   Function MessageBeep(Utype : Integer) : Integer; StdCall 'user32.dll';
Implementation

Begin
End.
```

This VPU library allows getting a new function named MessageBeep.
This function is not implemented in the VPU itself but in a DLL named User32.dll.
It is necessary to pass a parameter named Utype (Integer type)to the function.
StdCall allows specifying to the compiler that it must respect the standard mode for passage of parameters predefined by Microsoft for the 32Bits applications.

In case of MessageBeep, the function in User32.dll is also named MessageBeep.
Attention: respect scrupulously the case sensitive in the declarations
The following declaration is not correct:
Function MESSAGEBEEP(Utype : Integer) : Integer ; **StdCall** 'user32.dll';

To redefine the name of the function contained in a DLL

You can redefine the interface name of the procedure in the DLL. The following declaration allows it:

```
Unit
Interface
  Function Beep (Utype : Integer) : Integer; StdCall 'user32.dll' name 'MessageBeep';
Implementation
```

You define a new exploitable function in Visual I/O whch is Beep and corresponds to MessageBeep in user32.dll.

Localization of a DLL

It is possible that Visual I/O does not find the called DLL. Then you can copy your DLL into a Windows directory. For example, you can copy your DLL into C:\Windows. The Visual I/O runtime will locate this DLL.

You can also specify the complete path for the DLL (not recommended).

Function **Go**(Utype : Integer) : Integer ; **StdCall** 'C:\Proj1\Test.dll' name 'Start';

Visual I/O places many data types at the disposal so that the interface with DLL is the most compatible. However, the generally encountered problem is related to the character strings. In C language, for example, the type always used is PChar (Pointer on char, ended by the Null character 0). In Pascal, the character strings are different, and the first character is the length. To typecast (change the type for compatibility) you can use the PChar function from the Windows.pas library (source is included in Visual I/O).

Example to pass a text to a procedure in a DLL.

Example: **ExternFunc**(PChar('Hello')); PChar('Hello') returns the address of the first character in the string 'Hello'. The PChar function also ADD (if it is not the case) a Null character to the end of the text string.

Pointers and typecasting

Pointers

A pointer is a variable that its content is not a data but a memory address. The pointer utility is to access to a variable by its memory address. The pointer type doesn't exist in Visual I/O because it is equivalent to an Integer (32 bits). However, the pointer type is defined in Sysplc.Pas for simplification for reading.
Type Pointer = Integer;

Example :

```
Var
  I     : Integer;
  PInt  : Integer;
Begin
  PInt:=Addr(I);
```

PInt contains the memory address of the 'I' variable.
By convention, when one creates a pointer, the name of this pointer begins by "P" for a simplification of reading.

For access to a variable from his pointer, it is enough at first to get the address of the variable thanks to the function "Addr" (or the operator "@") that we saw previously.
Then in the second time, you can have access to the newly allocated memory by dereferencing the pointer; that is, use P^.

Example :

```
Begin
  PINT^:=10;
  PINT^:= PINT^+ 1;
Equivalent to
Begin
   I:=10;
   I:=I+1;
```

Memory allocation - GetMem and FreeMem

These 2 instructions are enabled to allocate memory and to release memory.

You can allocate (GetMem) a dynamic variable created and a pointer to the address of the block.
You dispose of a dynamic variable of a given size.

```
Var
  PTByte : Pointer;  //** Memory Pointer
  PTdest : Pointer;  //** Other pointer

Begin
  GetMem   (PTByte,80);  //* Creates a dynamic variable and a pointer to the
                         address of the memory block of 80 bytes size
  FillChar (PTByte,80,0);  //** Fills contiguous bytes with (80 bytes) with 0
```

Copy bytes from a source to a destination

```
  GetMem  (PTDest,200); //* Creates a dynamic variable and a pointer to 200 bytes
  Move    (PTByte,PTDest,80); // The Move procedure copys the first area in the
                              // second one
```

Dispose memory

```
  FreeMem  (PTint,80);   //** Release the first memory area (80 bytes)
```

GetMem : Store the address of beginning of the allocated memory in PTByte.
FillChar : Fill contiguous bytes from the address stored in PTByte with the specified value.
FreeMem : Release the memory block at the address stored in PTint and with a given size.

Typecasting

Typecasting allows changing or changing temporarily the type of a variable.
To typecast something, you can simply put the type of variable in front of the actual variable inside of parentheses.

Example :

```
  PTByte : Pointer   //** This type of pointer can point everywhere
```

To reach an element of the array of Byte, it is advisable to typecast the element to be able to reach its fields.

```
TArrayByte(PTByte)[1]:=10;
// We say in the compiler that the variable PTByte is the type Array of Byte.
```

Attention: `//** All the typecastes named Tarrayxxx begin from the indice 1`

TArrayByte is defined like this in Sysplc.pas :

```
Type
  TArrayInt  = Array [1..10] Of Integer;
  TArrayWord = Array [1..10] Of Word;
  TArrayByte = Array [1..10] Of Byte;
  TArrayBool = Array [1..10] Of Boolean;
  TArrayReal = Array [1..10] Of Real;
  TArrayDouble = Array [0..10] Of Double;
  TArraySingle = Array [0..10] Of Single;
  Pointer = Integer;
```

If we want to use the memory block PtByte as an array of Reals, it would be necessary to write as this

```
TArrayReal(PTByte)[1]:=345.67;
TArrayReal(PTByte)[2]:=PI;
```

The memory zone can be manipulated as. Example

```
Var
  B1 : Byte;
  R1 : Real;
Begin
  TArrayByte(PTByte)[1]:=10;   //*** Like an array of bytes
  B1:=TArrayByte(PTByte)[1];   //** B1 receives 10

  TArrayReal(PTByte)[1]:=345.67;  //*** Like an array of reals
  R1:=TArrayReal(PTByte)[1];      //** R1 receives 345.67
```

Retrieving the procedures and functions addresses

Address of procedures and functions are stored in system pointers. When the program calls a function, it makes an indirect

Example :
Variable pointer STRPAS contains the final address of the STRPAS procedure(of sysplc).
Variable pointer SETFILLCOLOR contains the final address of the SETFILLCOLOR procedure (system).

By writing

```
Var
 PTF,PTG : Pointer;
Begin
  PTF := ADDR (SETFILLCOLOR);
  PTG := ADDR (SETCOLOR);
  PTG^:=PTF^;  //*** The SETCOLOR pointer is modified with SetFillColor address
  SetColor(ClREd);  //*** Call SetFillColor now and not SetColor.
End;
```

PTF Contains the address of the variable pointing on the procedure SETFILLCOLOR.
PTG Contains the address of the variable pointing on the procedure SETCOLOR.
PTF^ Returns the beginning of the memory address of the SETFILLCOLOR procedure.

☞ **Note** : See also proceeding chapter Loading and unloading DLL manually

The Windows library contains the ModifyAddr procedure:

```
Procedure ModifyAddr( Var AddrDest; Var AddrSource );
```

Enable exchanging the address of a procedure with another one.

Example :

```
Begin
  PTF := ADDR (SETFILLCOLOR);
  PTG := ADDR (SETCOLOR);
  ModifyAddr (PTG,PTF);  //*** equivalent to PTG^:=PTF^;
  SetColor(ClREd);      //*** Call SetFillColor et not SetColor.
End;
```

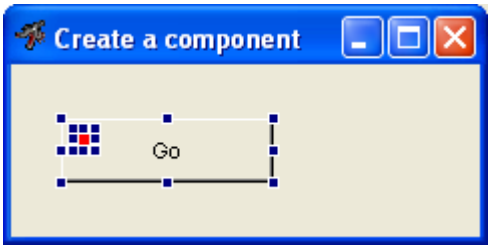
Creating Visual IO components

Visual I/O allows creating components that will be placed in the ToolBox.
How to create a component:

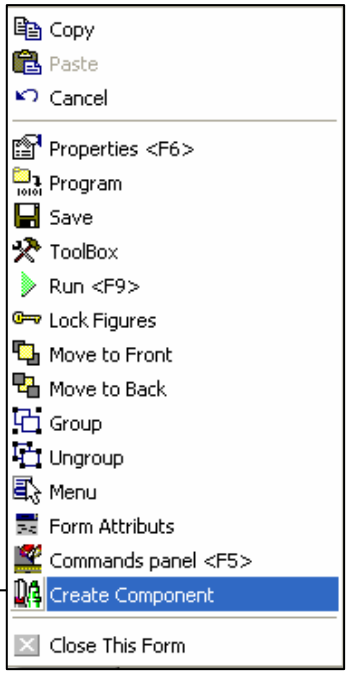
1/ Create your new component as a total part of the form graphically.
2/ Place behind the components of the animation (programs and functions).
3/ Run the form for final test.

At this step the graphic and program parts are ok. You can now copy the source code into the Visual I/O components editor.

Example of component - ButtonLed



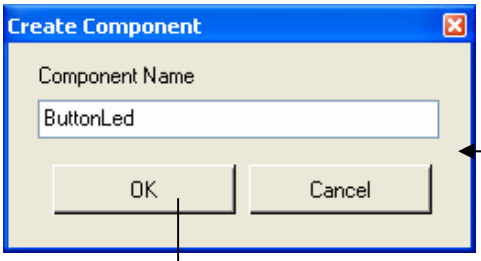
*After having drawn the component, **group it**.*
This one is composed of a button and a rectangle symbolising Led

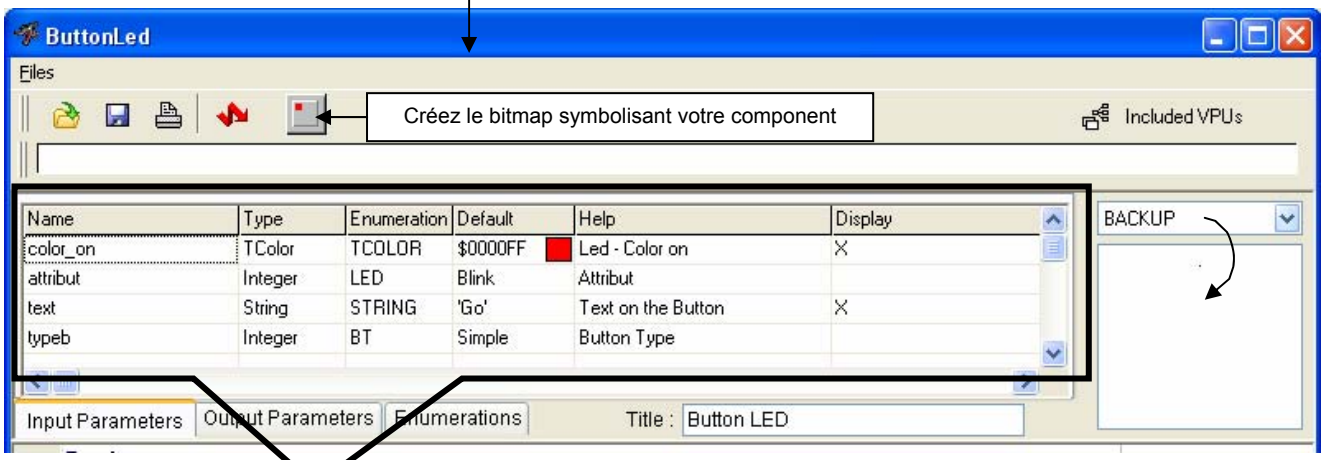


Click with the right button of the mouse, a floating menu appears.

Click on CreateComponent

A dialog box appears.
Write the name of the component.
*Here ButtonLed will create a compiled file **ButtonLed.VPU**.*





The top part allows defining the parameters that the programmer will have to be informed of when he will use the component in forms.

In the right hand side (listbox) allows listing all the necessary libraries for the component program.
Bring by Drag & Drop the library name in the combobox to the listbox below.

The **zone Title** receives the title of the component which will appear in the toolbox when the mouse rollovers the component.

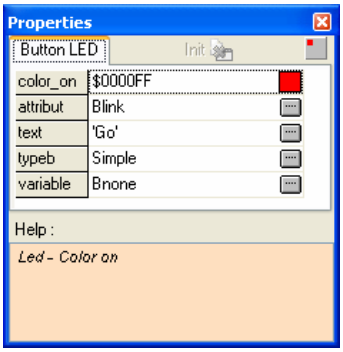
Input Parameters | Output Parameters | Enumerations | Title : Button LED

```
Begin
  If Start_Visu Then SetText(Text);
  If TypeB = 1 Then ToggleButton(Variable,-2)
               Else If Button then Variable:=Not(Variable);
  If Attribut=1 then
  Begin
    If BT_500=1 Then
    Begin
      Next_Group;
      If Variable Then
      Begin
        If GetFillcolor=Color_On Then SetFillColor($COCOCO)
                                 Else SetFillColor(Color_On);
      End Else SetFillColor($COCOCO);
    End;
  End Else
  Begin
    Next_Group;
    If Variable Then SetFillColor(Color_On)
                Else SetFillColor($COCOCO);
  End;
end;
```

The low part receives the source code applied to the first component drawn on the screen.
*To be able to control the other objects like the rectangle of the led, the **Next_Group** function will be used in the creation of components.*

Defining the user parameters

This window of properties appears at design time after placing a new component on the form.



Here 5 parameters are defined:

Color_On : To choose the color of the led. A palette appears by double-clicking on the cell.
Attribut : To display a list of options (Blink, Normal).
Text : To affect a variable (string) or a text written on the button.
TypeB : Allows choosing in a predefined list. Simple push-button or toggle button.
Variable : allow assigning a Boolean variable following the state of the button.

Variables and constants in input parameters

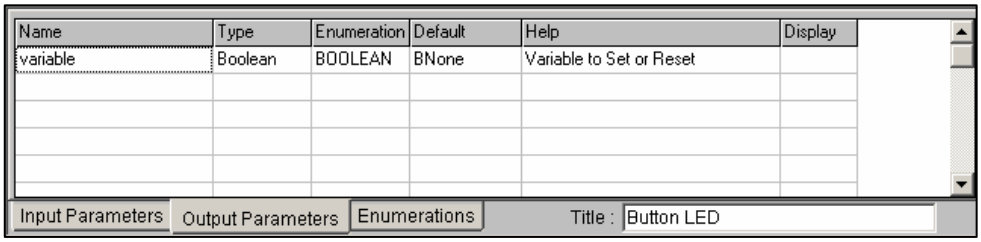
Name	Type	Enumeration	Default	Help	Display
color_on	TColor	TCOLOR	$0000FF	Led - Color on	X
attribut	Integer	LED	Blink	Attribut	
text	String	STRING	'Go'	Text on the Button	X
typeb	Integer	BT	Simple	Button Type	

Input Parameters | Output Parameters | Enumerations | Title : Button LED

Write the variable name, then double-click in the *type* column for choosing one of the **predefined types**. Choose a default value. Also, write a help text if necessary.

Color_On is *TColor* type, is enabled to display the palette of colors in the properties box.
Attribut is *Integer* type. However, the programmer can choose only default value. An enumeration is stipulated (Enumeration of the Led type, see below).
Text is a simple variable of *String* type or a constant String ' GO '.
TypeB is an *Integer* type and it is associated to an enumeration of the *BT* type

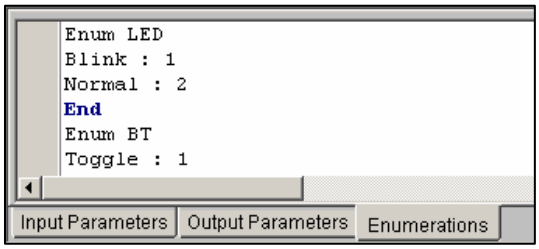
Variables only in output parameters



Here the *variable* parameter will receive only one existing Boolean variable in the list of the global variables. By precaution, the **BNone** variable is written by default. This variable is an existing System variable used for default assignments.

Enumerations

Declare your enumerations in the third tab of the editor.



The enumerations allow predefining values.

These predefined values will be displayed in a comboxbox in the future component properties box. Texts will be equal to a Number (view like a constant)

Example :

The reserved word **ENUM** is followed by the name of the enumeration.

ENUM …. **End**

The reserved word **END** closes the enumeration.

```
Enum LED
      Blink  : 1  // Here the texts Blink and Normal will appear in front of the variable Attribut
      Normal : 2
End
Enum BT
   Toggle : 1    // Here the texts Toggle and Simple will appear in front of the variable
   Simple : 2    // TypeB
End
```

The ButtonLed Program (sources provided in C:\API32\Modules)

```
Begin
  If Start_Visu Then SetText(Text);
  If TypeB = 1  Then ToggleButton(Variable,-2)
                Else If Button then Variable:=Not(Variable);
  If Attribut=1 then
  Begin
    If BT_500=1 Then
    Begin
      Next_Group;
      If Variable Then
      Begin
        If GetFillcolor=Color_On Then
        SetFillColor($C0C0C0) Else
        SetFillColor(Color_On);
      End Else SetFillColor($C0C0C0);
    End;
  End Else
  Begin
    Next_Group;
    If Variable Then SetFillColor(Color_On)
                Else SetFillColor($C0C0C0);
  End;
End;
```

To group shapes in a component

The beginning of the program always refers to the first shape drawn.
To reverse the order of the shapes, you can use **bring to front** and **put behind** commands in the editor.
To assign the program to other shapes of the component, you can use **Next_Group** or **Previous_Group**.
Another useful function is **TypeFig** that returns the type of the shape (CLINE, CBMP etc.) to know the values returned by this function and to see the constants defined in Windows.PAS.

Commutation from a shape to another in a component.

The **Go2** procedure cannot be used without knowing the exact name of the shapes.

Let us consider that a component contains 3 Circles, with each use of this component in synoptic. The system creates 3 new distinct circles with 3 names (CircleX, CircleX+1, CircleX+2). So we can't code the names of the circles into the program of the component. However it is possible to know the name of these circles included in the component at Runtime.

Example :
Get the name of the 3 circles at initialisation time, and then work with the instruction **Go2** to commut from one circle to another.

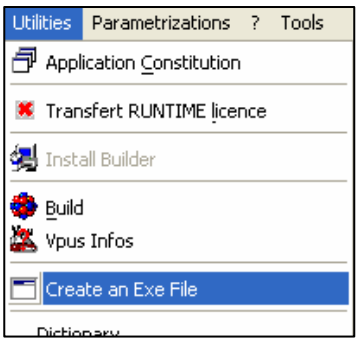
```
Var
  Name1,Name2,Name3 : String ;
Begin
  If Start_Visu Then    //* The component name is stored in memory
  Begin
    Name1 :=GetFigname; //* Get the name of the First Circle
    Next_Group ;
    Name2 :=GetFigname; //* Get the name of the Second Circle
    Next_Group ;
    Name3 :=GetFigname; //* Get the name of the Third Circle
  End;
  Commut_Fig('Name2') ; //* Go to on the second circle
  SetFillColor(Color2);
  Commut_Fig('Name3') ; //* Go to on the third circle
  SetFillColor(Color3);
  Commut_Fig('Name1') ; //* Go to on the first circle
  SetFillColor(Color1);
End;
```

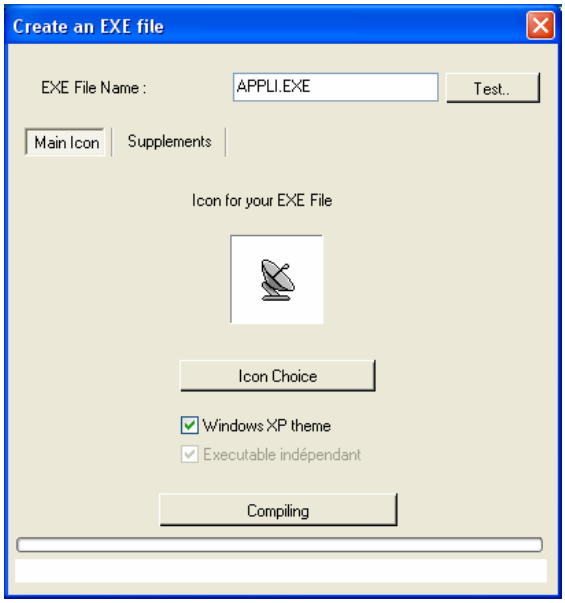
Create an final executable file

Contrary to previous version, visual I/O 7 creates executables files (.exe), including all the necessary data (Bitmap, menu, files of configuration) to your applications.

The program generated in every launching of the application by the button or by the **<F9>** key is By default named APPLI.EXE.

This name as well as various parameters is modifiable in the menu “utilities” and the “creates an executable” item.

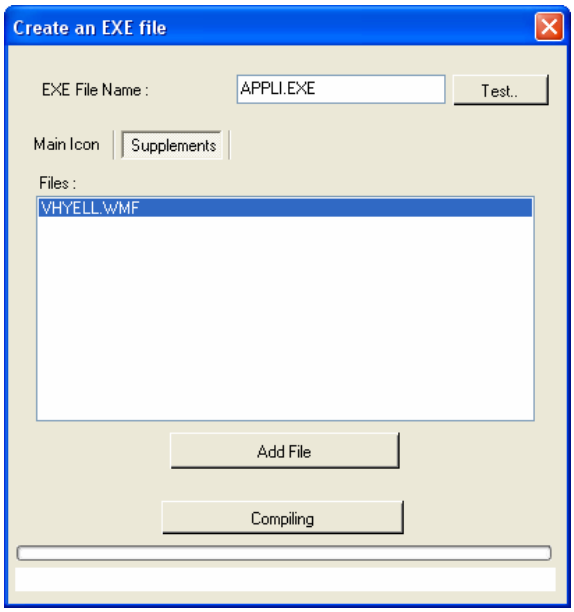




Executable File name To specify the executable name generated at compilation time or at each launching from the editor.
Button Icon Choice To choose an icon for your application.
Windows XP Theme To validate the new XP looking for your application. It is available only under Windows XP.

indépendant Executable: Not used.
Compiling :To create the final Executable file name specified in the Executable file name edit box.

Supplements Tab



Files
Contain the list of the files that have to be added to the executable (in the resources), besides files traditional as forms and other used bitmaps PDF or html files.

Add File Button To navigate for choosing a new file.

The program ***SCANRESEXE.exe*** is enabled to display the resources in the executable generated by Visual I/O.
See Menu *Tool/add tool* or *Upgrading*

The ***ExtractRes*** function is enabled to extract a resource file from an executable by program.

Debugging a program

Whatever care is brought to the writing of your program, it will always contain some errors, or bugs, which will prevent it for working as well as expected. The debugging consists in looking for these errors and repairing them. The IDE includes an integrated debugger which helps you to detect and to correct the errors contained in your programs. The functionalities are:

- Controlling the program execution
- Watching the variables values and structures elements
- Modifying the data values at debug time.

To debug a program, it is necessary to launch it in debug mode.
The program is under the control of the editor, but it will be slowed down because the code generated remains the same.

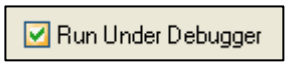
Step 1 – The breakpoints

Compile the program or the library. Place the break points to the lines.

```
{*** Button1 ****************************************}
SObject Procedure Button2;
Var
 HP       : Integer;
 DSocket  : Integer;
 TB       : Boolean;
 Buf      : Array [0..100] Of Byte;
 I        : Integer;
Begin
  if Button Then
  Begin
     TB:=SocketOpenConnection('192.168.0.1',80,DSocket);
     I:=SocketSendBuf ( DSocket , Buf, Sizeof(Buf));
     I:=SocketReceiveLength ( DSocket);
     SocketCloseConnection (DSocket);
  End;
End;
```

Here a break point is placed on the line which opens a TCP/IP connection. As long as the operator does not click the button, the program will not execute the program in the Begin End block. When the program execute the SocketOpenConnection instruction, the program will stop and the focus will be given to the editor allowing going step by step in the program and showing the different variables of the program.
To continue the program, press the <F9> key.

Step 2 – Run under debugger



Launch the program from the editor with the checkbox Run under Debugger check.

The program is compiled normally without modification. Only this program is launched under a specific mode in Windows. It is the Debug mode that allows placing breakpoints and displaying the variables.

Step 3 – The program stop at the break point

When the program arrives at a break point, it stops and gives the hand to the editor displaying a little windows (only the first time) then the line in highlighting totally in red.

A window indicates in the editor where is the break point.
To move Step by Step strike **<F7>**
To continue normally strike **<F9>**.

The line on which the processor is stopped is highlighted.
Attention: this line is not executed yet.

```
if Button Then
Begin
    TB:=SocketOpenConnection('192.168.0.1',80,DSocket);
    I:=SocketSendBuf ( DSocket , Buf, Sizeof(Buf));
    I:=SocketReceiveLength ( DSocket);
    SocketCloseConnection (DSocket);
End;
```

Move step by step <F7> key

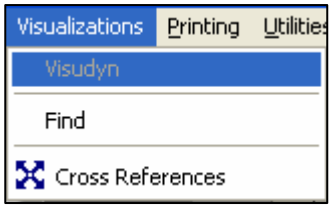
If another editor is open and the SocketOpenConnection function is visible, the editor will go to the first line in step-by-step mode, otherwise the program goes to the line.

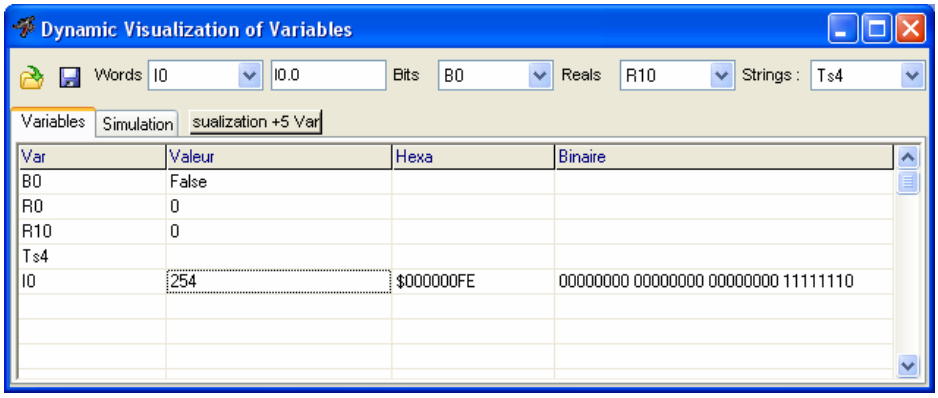
```
Begin
    TB:=SocketOpenConnection('192.168.0.1',80,DSocket);
    I:=SocketSendBuf ( DSocket , Buf, Sizeof(Buf));
    I:=SocketReceiveLength ( DSocket);
```

TB = True

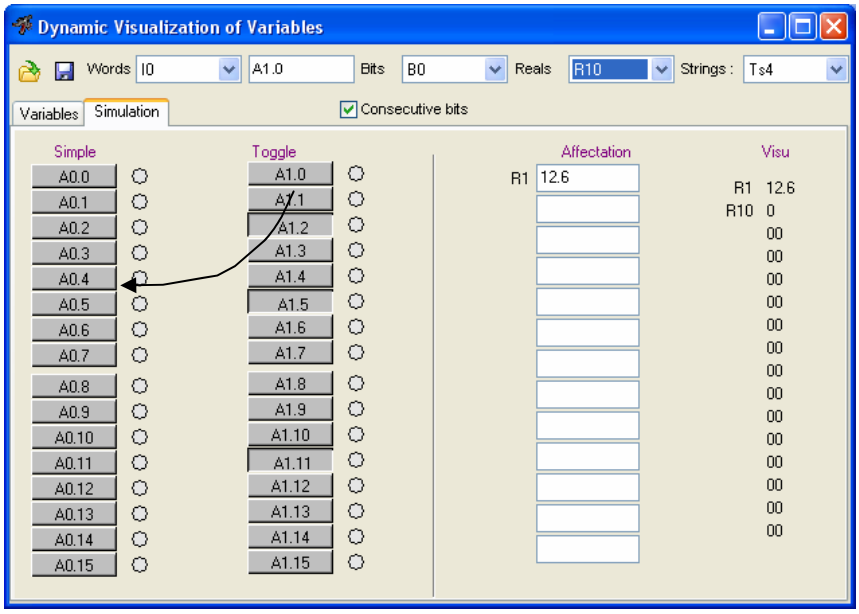
Watching a variable

Place the mouse on the variable. If it is visible and a little window exits, the little window will be displayed with its value.
Above the content of the TB, variable is displayed.
To display the global variables, click in the main menu Visualizations and visudyn.





Only a grid of visualization of the global variables is displayed. The column value, Hexa and Binary allow visualizing and forcing the value of the global variable.



.

Simulation makes it possible to affect simple or toggle pushbuttons on the global variables

Bring by drag & drop the Boolean variable from the combobox or editbox to the button, the editbox or the visualization (Visu).

The check box *consecutive Bits is* enabled to fill the consecutive buttons automatically with the consecutive bits.

Avanced programming

The assembler language

From version 7, visual I/O allows programming directly in assembler language. It is necessary to know a minimum of the processor operation to be able to approach this part.

The assembler is the low level programming language. It means that it is very close to the material. Every instruction corresponding to a machine code can be understood by the microprocessor. The assembler code is the legible version of the machine code.

If it is nowadays unthinkable to program an application in assembler completely (so as purposes of learning naturally), numerous situations will make that we have to have appeal to this language. For example, it is the case when the speed is a critical factor, and when the slightest microsecond is precious.

Example of assembler programming

The parameter Range: integer is contained in the register EAX and the result (Result) is sent back to the same register EAX.

```
{ ->EAX      Range
{ <-EAX      Result}

Var
  RandSeed : Integer; //** Variable used in the random function
//**** Generate a random integer
Function Random( Range : Integer) : Integer;
Asm
  Asm
     IMUL    EDX,RandSeed,08088405H  //** Multiplication of ranseed
     INC     EDX
     MOV     RandSeed,EDX     //** Get the external Ranseed variable Ranseed
     MUL     EDX
     MOV     EAX,EDX          //** Final result sent to EAX
  End;
End;
```

Here the Random function doesn't start by Begin but by **ASM**.
That allows not affecting the registers which do not used, by entering the Random function.

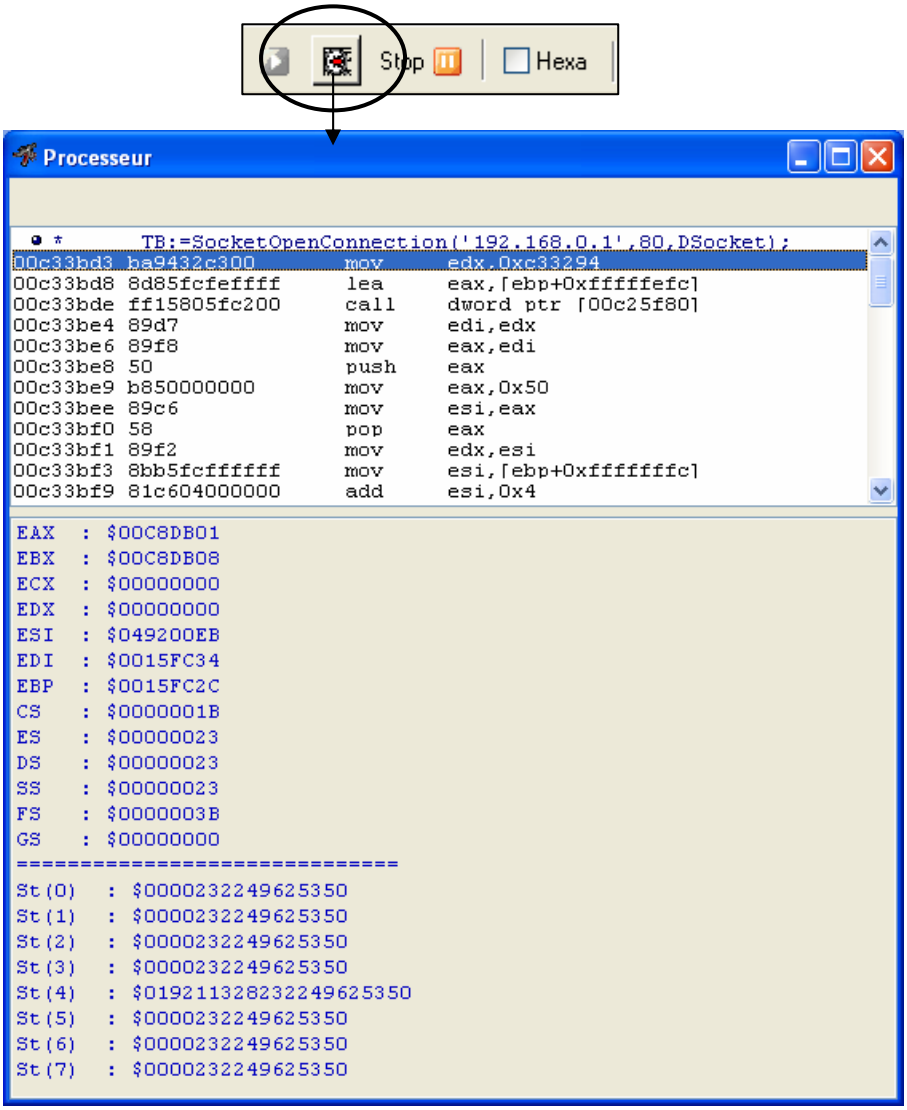
```
{** Bytes Array Comparison - Returns Address different or 0 *
Function  CompareBlockEx ( Var B1,B2 ; Nbyte : Integer) : Integer;
Begin
 Asm
   PUSH ESI
   PUSH EDI
   MOV ESI,[EAX]  //** EAX contains the address of B1
   MOV EDI,[EDX]  //** EDX contains the address of B2
   MOV AL,0
   CLD
   REPE CMPSB     //** REPE use ECX containing the NByte value
   SETE AL
   CMP AL,1
   JZ  @OK
   MOV EAX,ESI
   JMP @FINI
@OK:              //** label OK
   MOV EAX,0
@FINI:            //** Label FINI
   POP EDI
   POP ESI
 End;
End;
```

☞ **Note** : In *Run under debugger* mode, each assembler line can be executed in step by step.
Watching the processor registers is also possible.

Watching the processor registers

At every break point or to move single step in debug mode, it is possible to show the processor registers and the arithmetical coprocessor. It is also possible to show the equivalent in assembler of the line code in Pascal. It still demonstrates that visual I/O is not an interpreted system, but a well compiled system.

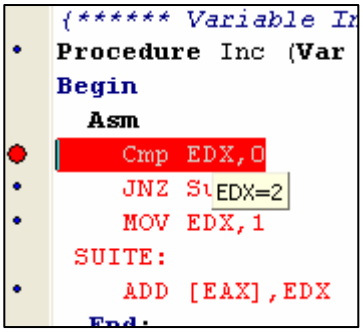
When the break point is made, click the button to display the processor.
A window appears, showing the code disassembled, the various Registers of processors as well as the registers of the arithmetical coprocessor.



In the main menu, the button Stop also allows suddenly stopping the application in debug.
The checkbox *Hexa* allows showing the numerical values in hexadecimal or in decimal.

In assembler programming, it is possible to show the processor registers by moving over the source code with the mouse.

In the case of **INC** procedure in Sysplc.pas,
In step-by-step mode, you can display the differents Registers of the processor used by this procedure.



StdCall call convention

Parameters are transferred to procedures and functions via CPU registers or the stack, depending on the routine's calling convention. The procedures and functions of Visual I/O have the REGISTER convention. The parameters are passed by the processor registers (EAX,EDX,ECX) from left to right.
If the routine has more than three parameters, then the parameters are sent by the stack.

The **STDCALL** convention pushes parameters into the stack in the reverse order in which the compiler encounters them in the actual parameter list. Passing parameters are made from right to the left.
The Windows API uses the "STDCALL" to call convention.

Interface with external DLL.
Callback procedure used by Windows.

Example of use

Windows.pas interface library with the system User32.DLL and the function ShowWindow that allows showing or hiding a window of a given handle.
Here 2 parameters are passed via the stack from right to left (nCmdShow puis *hWnd*).

Function ShowWindow(**hWnd**: Integer; **nCmdShow**: Integer): Boolean; **STDCALL** 'User32.dll' name 'ShowWindow';

The library RICHINT.PAS contains a function call by Windows named StreamSave. Windows will call this function by passing four parameters via the stack from right to left.
This function must restore the context of the Visual I/O application (variables) and clean up the stack before returning to Windows.

```
//**** Call Back Proc Called from Windows ************************************
Function StreamSave(dwCookie: Longint; pbBuff: Integer;
                  cb: Longint; var pcb: Longint): Longint; STDCALL;
Var
  FIC : Integer;
begin
  Result := NoError;
  Fic:=dwCookie; //** Handle of the Fileclose
  Filewrite(fic,pbBuff^,cb);
  pcb := 0;
end;
Begin
  I:=@(StreamSave);                //** StreamSave function address
  EditStream.pfnCallBack := I^;   //** structure sent to windows.
End;
```

☞ **Note**: It is also possible to create internal functions in visual I/O with the StdCall convention.
These will also work normally, but can be less quickly due to the mechanism of variables Push and Pop.

Send windows messages

The windows controls run with the reception of messages coming from an application.
For example, to show or hide a window, it is possible to use the Showcontrol or Hidecontrol procedure but also the sending of a message to the window, thanks to the **SendMessage** function and parameter **SW_HIDE** or **SW_SHOW.**

The GetMainHwnd function returns the form handle.
The GetHWnd function returns the control handle (Listbox, Combobox etc).

```
SendMessage (GetMainHwnd, SW_HIDE,0,0);        //* to hide the form
SendMessage (GetMainHwnd, SW_SHOW,0,0);        //* to show the form
```

Windows SDK

```
LRESULT SendMessage(
    HWND hWnd,          // handle of destination window
    UINT Msg,           // message to send
    WPARAM wParam,      // first message parameter
    LPARAM lParam };    // second message parameter
```

Control in totality Windows

Windows is based on the sending and management of messages sent to the application.

Why messages ?

One of the keys to traditional Windows programming is handling the **messages** sent by Windows to applications. Generally speaking, a message is some information sent from one place to another. An event is usually generated in response to a Windows message sent to an application.
However, someday we will want to process some uncommon messages like CM_MOUSEENTER, which happens (is posted by Windows) when mouse cursor enters the client area of some component (or form).

Handling messages on our own requires a few extra programming techniques.

For example: You press the left button of the mouse and a message WM_LBUTTONDOWN is automatically sent by Windows. Your application can be warned of this action in several ways.

Notably by the events managed directly by the core of the program generated by Visual I/O which are automatically started when this message is intercepted by your application.
You will test the actions with the available procedures (IS_ME, ARRIVED ALLOCATES etc.) or by the event programming that is a subtle mixture of generation of messages and reactions in front of these messages.

Principle

Any possible action under Windows is associated with a message. This message has a "message constant" who allows identifying it (see Windows.pas or Microsoft SDK). Other parameters are also associated to the message which allows getting back the position of the mouse at the click time.

Structure of a message defined in Windows.pas:

```
TMessage = Record
   Msg    : Integer; //** The Msg field represents the Windows Message code.
   WParam : Integer; //** First parameter
   LParam : Integer; //** Second parameter
   Result : Integer; //** The Result field holds the return value.
End;
```

What happens when a message is sent?

An event occurs (click a button of the mouse, press a key).

Windows in the form of a message of **TMessage** type translates this event. Windows generates ONE OR SEVERAL messages depending on the case. For example, in the case of pressing key, Windows sends WM_KEYDOWN and WM_KEYUP when we release the key (non-system key).
Most of the messages are placed in the stack of the application messages then sent.
The core of the Visual I/O application processes these various messages, but also has the possibility to send them to the user application.

The application scans this stack permanently and acts according to the message. So the application can get various manners. Having got the message of **TMESSAGE** type, it has access to all the parameters and the information concerning it.
The program generated by Visual I/O also allows passing these messages on you even before these messages have already been treated by the program. You can then dispatch or call specific procedures according to the received messages.

Small Windows messages handler

The ReceiveMSG procedure receives the messages

```
Uses Windows;
//***** Receive Messages From Windows ********************
Procedure ReceiveMSG ( Handle : Integer; Var Msg : TMessage);
BEGIN
  MSGCount:=MSGCount+1;
  IF Msg.MSG=WM_LBUTTONDOWN Then Addstring('WM_LBUTTONDOWN') Else
  IF Msg.MSG=WM_LBUTTONUP   Then Addstring('WM_LBUTTONUP')   Else
  IF Msg.MSG= WM_MOUSEWHEEL Then Addstring('MOUSE WHEEL');
End;
```

The DispatchMes procedure enables adding a supplementary callback handler for the windows messages.

```
{*** Button1 ************************************
SObject Procedure Button1;
var
 i : Integer;
Begin
  if Start_visu Then
  Begin
    I:=Addr(ReceiveMSG);  //*** I is the procReceiveMsg pointer
    Dispatchmes(I^);   //** Pass the contents of the pointer (effective address)
  end;
End;
```

Filtering of the messages

Thanks to the handle, you can know whether the message concerns a particular control (listbox, combo, window, radio button, grid.).
For example, we test only the clicks in the listbox1

```
Var
  ListBoxHandle : Integer;

//***** Receive Messages From Windows ********************
Procedure ReceiveMSG (Handle : Integer; Var Msg : TMessage);
Begin
  MSGCount:=MSGCount+1;
  If ListBoxHandle=Handle Then
  Begin
    Commut_Fig('ListBox1');
    IF Msg.MSG=WM_LBUTTONDOWN Then Addstring('WM_LBUTTONDOWN') Else
    IF Msg.MSG=WM_LBUTTONUP   Then Addstring('WM_LBUTTONUP')   Else
    IF Msg.MSG= WM_MOUSEWHEEL Then Addstring('MOUSE WHEEL');
  End;
End;

{*** ListBox1 ************************************
SObject Procedure ListBox1;
Begin
  If Start_visu Then ListBoxHandle:=GetHwnd; //** Get the listbox handle
End;
```

Bonus32 library

This library offers treatment facilities on character strings.

Function Right `(S :String) : String;` *`//Retrieve the last character in a String;`*

Returns the character that appears at the end of a string. Example Right('ARSOFT') returns 'T'

Function CompareText `(S1, S2: String): Integer;` *`//*** Compare 2 strings`*

`Compares two strings by ordinal value without case sensitivit.`
CompareStr compares S1 to S2, with case-sensitivity. The returned value is less than 0 if S1 is less than S2. The returned value is 0 if S1 equals S2. The returned value is greater than 0 if S1 is greater than S2.
`CompareText('ARSOFT','arsoft')` `// retruns 0 because the string are equivalents.`

Function BuildString

```
(StrBase : String;
SubStr1,SubStr2,SubStr3,SubStr4 : String) : String;
```

Build a character string according to a format and parameters.
Parameters are inserted into the formatted character string. < SubStr1 > will replace the character %1. < SubStr2 > will replace the character %2, etc. The number of specified parameters must be identical or superior among parameters used in the character string < StrBase>. If the character string <Format> contains more parameters, the supplementary parameters will be replaced by empty strings.
Use of the Buildstring function in the multi-languages applications

If your application contains messages shown in several languages, the function BuildString allows replacing the optional characters in all the managed languages.

For example :
`TSFR:='Utilisateur %1 ne peut pas accès au fichier %2'`
`TSUK:='File %2 cannot be reached by the user %1'`

Function FormatString `( StrBase : String; Attributs : Word) : String;`

Format a character string according to the chosen options. The Combination of these options is possible.
`StrUpper, StrLower, StrBlank , Straccent .`
`For example:`

```
FormatString('Procedure',StrUpper And StrAccent) returns 'PROCEDURE'
```

Function StringOccurrence

```
( StrBase : String; Substring : String;
  Nocasesensitive : Boolean) : Integer;
```

Calculate the number of occurrences of a specific character string (by respecting the research criteria) in another character string.

```
Res = StringOccurrence ('anastasia', 'a');          // Returns 4
Res = StringOccurrence ('toto titi tito', 'to');    // Returns 3
Res = StringOccurrence ('grille pain',  'X');       // Returns 0

StringOccurrence ('Le merle est dans le chêne', 'le')        // Returns 2
StringOccurrence ('Le merle est dans le chêne','le', True);  // Returns 1
StringOccurrence ('Le merle est dans le chêne','le', False); // Returns 3
```

Function StringComplete `(StrBase : String; Size : Integer; Complement : String) : String;`

Send back a specific character string of a given size. This string can be truncated or completed by spaces (or by another character) to reach the required size.

```
Complete('ARSOFT', 8)          // Renvoie 'ARSOFT  '
Complete('Arsoft', 8, "A")     // Renvoie 'ArsoftAA'
Complete('Arsoft', 4)          // Renvoie 'Arso'
```

Function ControlLoadFromFile `(FileName : String; CtrlName : String; Option : String) : Boolean;`

Load an EditBox, Listbox, combobox or Data grid with the contents of a file.

Caseof the listbox or combobox:
These 2 controls are cleared and loaded by the contents of a text file.
Case of the Edit box.
If multi lines, the loading is performed as a listbox(line per line).
If mono line (simple editbox), only the first line of the file is loaded.

Filename : File name to be loaded in the control..
CtrlName : Control name (Name of the graphical component)
If no name, the current component is used, else a Go2 procedure is performed.
Option : Text String containing the option (ex separator in a grid).

Example : ControlLoadFromFile('NOTICE.TXT'); *//Loading the current component*
ControlLoadFromFile('NOTICE.TXT','LISTBOX1'); *// Load in Listbox1*
ControlLoadFromFile(PARAMETERS.CFG','WINDOW1',';'); *//load in the Window1 grid with the separator ; (csv format)*

Function ControlSaveToFile `(FileName : String; CtrlName : String; Option : String) : Boolean;`

Save the contents of an EditBox, Listbox, combobox or datagrid in a file.

Caseof the listbox or combobox:
The contents of the two controls are saved in a text file.
Case of the Edit box.
If multi lines, all the lines are saved in the file.
If mono line (simple edit box), only the first line is saved in the text file.

Filename : File name of the control contents to be saved.
CtrlName : Control name (Name of the graphical component)
If no name, the current component is used, else a Go2 procedure is performed.
Option : Text String containing the option (ex separator in a grid).

Example : ControlSaveToFile('NOTICE.TXT'); *// Save the current component*
ControlSaveToFile ('NOTICE.TXT','LISTBOX1'); *// Save the Listbox1*
ControlSaveToFile (PARAMETERS.CFG','WINDOW1',';'); *// Save the grid contents with the separator ";" (Creating a CSV format compatible with Excel)*

Function GetHandle `(CtrlName : String) : Integer;`

Retrieves the handle of a Windows control (Window, Listbox, Combo). The handle of a control is often necessary when using the Windows API (For example, see Richedit control).

Bonus1 library

Function DiskInDrive `(Drive: String): Boolean;`

Indicates whether a disk is in the specified drive. No case is sensitive.

Example: Diskindrive ('a');

Gestfile library

This library manages the files in direct access or binary files. These instructions **cannot be used with** the text files seen before.

```
Const
  FmRead      = $80000000;
  FmWrite     = $40000000;
  FmReadWrite = $C0000000;
```

Function FileOpen `( FileName : string; Mode: Integer): Integer;`

Opens a specified file using a specified access mode. You can use FileOpen to open a file and obtain a file handle. The access mode value is constructed by or in one of the fmOpen constants with one of the fmShare constants defined. If the return value is 0 or greater, the function was successful and the value is the file handle of the opened file.

Var

HDLE : Integer;

Example :

```
HDLE:=FileOpen('DATABIN.BIN', FmReadWrite);
If Hdle>0 then FileRead (Hdle,Buf,1024);
```

Function FileCreate `( FileName : string): Integer;`

FileCreate creates a new file with the specified name. If the returned value is positive, the function was successful and the value is the file handle of the new file. A returned value of -1 indicates that an error occurred.

```
HDLE:=FileCreate('DATABIN.BIN');
If Hdle>0 then FileWrite (Hdle,Buf,1024);
```

Procedure FileClose `(Handle : Integer);`

FileClose closes a file given its handle. The handle is obtained when the file is opening with FileOpen or FileCreate.

```
HDLE:=FileOpen('DATABIN.BIN', FmReadWrite);
…..
If Hdle>0 then Fileclose(Hdle);
```

Function FileDelete `( FileName : String) : Boolean;`

DeleteFile deletes the file named by FileName from the disk. If the file cannot be deleted or does not exist, the function returns False.

```
If Filedelete('TEST.BIN') Then Message('File deleted'); // Message dans sysplc
```

Function FileRead `(Handle : Integer; var Buffer; Count : Integer): Integer;`

FileRead reads Count Bytes from the file specified by Handle into the buffer. The Count parameter indicates the size, in bytes, of the buffer. The function result is the actual number of read bytes, which may be less than Count. The Handle that is passed to FileRead must be opened with FileOpen or FileCreate.

```
ActualRead := FileRead(MyFileHandle, Buffer, SizeOf(Buffer));
```

Function FileWrite `(Handle : Integer; var Buffer; Count : Integer ): Integer;`
FileWrite writes Count Bytes to the file given by Handle from the buffer specified by Buffer. Handle is a file handle returned by the FileOpen or FileCreate method.The returned value is the number of bytes actually written, or -1 if an error occurrs.

```
HDLE:=FileOpen('DATABIN.BIN', FmReadWrite);
If Hdle>0 then FileRead (Hdle,Buf,1024);
```

Function FileSeek `( Handle, Offset: Integer): Integer;`
Use FileSeek to reposition the read/write point in a file that was opened with FileOpen or FileCreate. Handle is the file handle that is returned by FileOpen or FileCreate.

```
FileSeek(MyFileHdle,FileSize(MyFileHdle));
```

Function FileSize `( Handle : Integer) : Integer;`
Call FileSize to determine the size of the file specified by the file variable Handle.
To use FileSize, the file must be open. If the file is empty, FileSize(F) returns 0.

```
FileSeek(MyFileHdle,FileSize(MyFileHdle));
```

Function CopyBlocksToPos `(Source,Destination : String;`
`Position :Integer) : Boolean;`

Copy a part of a source file from beginning upto the specify position (excluded) into the destination file.

```
// Copy the beginning of the file
CopyBlocksToPos (FileNameHisto,SaveFile,PosFile);
// Copy the end of the file
CopyBlocksFromPos(FileNameHisto,'Tempx.dbf',PosFile);
```

Function CopyBlocksFromPos `( Source,Destination : String;`
`Position : Integer) : Boolean;`
Copy a part of a source file from the specify position (included) up to the end of this file into a destination file.

```
// Copy the beginning of the file
CopyBlocksToPos (FileNameHisto,SaveFile,PosFile);
// Copy the end of the file
CopyBlocksFromPos(FileNameHisto,'Tempx.dbf',PosFile);
```

Instructions for directories

Function DirectoryCreate`(NameDir : String) : Boolean;`
Creates a new directory. The returned value is True if a new directory is successfully created, or False if an error occurrs.
`DirectoryCreate('C:\MyProj\Datas\Files');`

Function DirectoryRemove`(DirName : String) : Boolean;`
To remove the directory specified by the Dir parameter. The returned value is True if a new directory is successfully deleted, False if an error occurrs. The directory must be empty before it can be successfully deleted.

Function ChangeDirectory`(DirName : String) : Boolean;`
Changes the current directory to the path specified by Dirname. If Dirname includes a drive letter, the current drive is also changed.

Function GetCurrentDir `: String;`
Returns the current directory.

Function DirectoryExist `( DirName: string) : Boolean;`
Call DirectoryExists to determine whether the directory specified by the Name parameter exists. If the directory exists, the function returns True. If the directory does not exist, the function returns False.

Function FileExtractName`( FileName: string): string;`
Extracts the name and extension parts of a file name.
`Ex : FileExtractName ('C:\API32\VPLC.EXE'); // Returns VPLC.EXE`

Function FileExtractPath`( FileName: String): String;`
Returns the drive and directory portions of a file name.
`Ex : FileExtractPath ('C:\API32\VPLC.EXE'); // Returns C:\API32\`

Function FileExtractExt `( FileName: string): string;`
Returns the extension part of a file name.
`Ex : FileExtractExt ('C:\API32\VPLC.EXE'); // Returns .EXE`

Function ProgRunning `( PName : String) : Boolean;`
The return value is true if the specified program is running. PName must contains the complete path of the program.
PNAME='APPLI.EXE' returns True and PNAME='C:\WORK\APPLI.EXE' could be return False.

RichInt library– RichEdit Control Interface

This library contains all the interface procedures with a richedit control. For using a text editor you can load the screen model **Richeditmain.VPK** or the Visual IO example.
For a first look, refer to the chapter *Richedit – Text editor for Windows.*

Add a new line.

Function RichAddstring `( Value : String; Handle : Integer ) : Integer;`
Ex: `RichAddstring('Visual I/O – Visual Pascal',GetHandle('Window1'));`

Delete all text from the rich edit control

Procedure RichClear `( Handle : Integer);`

Set the text of the control to the text in the specified buffer.

Procedure RichSetText `( PTText : Integer; Handle : Integer );`
Is set to the text in the buffer pointed by the PTText parameter. PTText must point to a null-terminated string
For example :

```
FileRead(Fic,Buf[0],1024);
RichSetText(Addr(Buf[0]),HDR);
```

Return the lines numbers in the richedit control.

Function RichGetCount `( Handle : Integer) : Integer;`

Return the number of characters present in the richedit.

Function RichGetTextlen `( Handle : Integer) : Integer;`

Insert a text string in front of a specific line. First line is index 0

Procedure RichInsert `( Index: Integer; S: string; Handle : Integer );`

Load a RTF File in a richedit.

Procedure RichLoadFromFile `( Filename : String; Handle : Integer);`

Save the contents of the richedit into a file.

Procedure RichSaveToFile `( Filename : String; Handle : Integer);`

Copy/Paste Commands

Remove the selected text from the edit control.

Procedure RichClearSelection `( Handle : Integer);`

Copy the selected text in the edit control to the Clipboard in CF_TEXT format

Procedure RichCopyToClipboard `( Handle : Integer);`

Copy the selected text to the Clipboard in CF_TEXT format and then delete the selection.

Procedure RichCutToClipboard `( Handle : Integer);`

Paste the contents of the Clipboard into edit control, replacing the current selection.

Procedure RichPasteFromClipboard `( Handle : Integer);`

Back out all the changes in the undo buffer.

Procedure RichUndo `( Handle : Integer);`

Select all the text in the Richedit.

Procedure RichSelectAll `( Handle : Integer);`

Text Attributes

Set or unset the protected attribute for the richedit contents

Procedure RichSetProtected `( Value: Boolean; Handle : Integer);`

Change the selected text color.

Procedure RichSetColor `( Value: TColor; Handle : Integer);`

Change the height of the text font (in pixels) of the selected text.
Procedure RichSetFontSize `( Value: Integer; Handle : Integer);`

Change the font name of the selected text.
Function RichSetFontname `( Value: String; Handle : Integer) : Boolean;`

Change the font attributs of the selected text.
Procedure RichSetFontStyle `( Value: Integer; Handle : Integer);`
One of these constants or combinaisons of constants.

```
CFM_BOLD            = $00000001;
CFM_ITALIC          = $00000002;
CFM_UNDERLINE       = $00000004;
CFM_STRIKEOUT       = $00000008;
```

Select text in Bold + underline :
Ex : `RichSetFontStyle (  CFM_BOLD Or  CFM_UNDERLINE, HDR);`

Return the selected text font attributes.
Return a value or a combination of values.
Function RichGetFontAttributes `( Handle : Integer) : Integer;`

Texts alignments

Procedure RichSetAlignment `( Value: TAlignment; Handle : Integer );`

```
Sets the alignment of the selected text. Value can take one of these 3 Values :
      taLeftJustify       = 0; //** Text is left-justified
      taRightJustify      = 1; //** Text is right-justified
      taCenter            = 2; //** Text is centered in the Richedit
```

Specify the formatting information for the current paragraphs.
Specify whether the paragraph is one paragraph of a set of bulleted ones
Procedure RichSetNumbering `( Value: TNumberingStyle; Handle : Integer);`
nsNone No numbering on the left.
nsBullet Bullets on the left with sufficient indentation for the bullets.

Return information on the selected text.
The returned value is one of these nsNone or nsBullet.
Function RichGetNumbering `( Handle : Integer) : TNumberingStyle;`

Specify the indent, in pixels, of the first line of the paragraph relative to the left margin.
Procedure RichSetFirstIndent `( Value: Integer; Handle : Integer );`

Return the indent, in pixels, of the first line of the paragraph relative to the left margin.
Function RichGetFirstIndent `( Handle : Integer)  : Integer;`

Return the indent, in pixels, of the paragraph relative to the left margin.
Function RichGetLeftIndent `( Handle : Integer)  : Integer;`

Specify the indent, in pixels, of the paragraph relative to the left margin.
Procedure RichSetLeftIndent `( Value: Integer; Handle : Integer );`

Searching

Searches a given range in the text for a target string
Function RichFindText

```
( SearchStr: string;  StartPos, Len: Integer;
  Options: TSearchTypes; Handle : Integer): Integer;
```

Use FindText to search a range of text in a rich edit control for the string specified by the SearchStr parameter. Only the text in the range starting at the position StartPos and continuing through the next Length positions will be searched. Use Options to specify whether the search should match whole words only and whether the search should be case sensitive. FindText returns the position of the search string, where 0 is the position of the first character in the rich edit control (as opposed to the first position in the search range). If the search string does not appear in the indicated range, FindText returns -1.
Use FindText rather than programmatically searching through the text of the control to keep from being thrown off by the encoding of rich text characteristics
Options can take the following values:

```
stWholeWord = 0;  //** All the words
stMatchCase = 1;  //** Case sensitive.
```

Sysplc library

This library is based on the windows API (Procedure and function available in the Windows system DLL). Not all the procedures are described, only the most interesting are commented.

Play a waveform equivalent to SystemHand sound.

Procedure Beep;

Display a specified message to the user. Window without caption.

Procedure Message `(Text : String);`

Use MessageEx to display a generic dialog box, a message and one button. Caption is the caption of the dialog box.

Procedure MessageEx `( Text,Caption : String);`

The message box contains two push buttons: Yes and No.

Procedure MessageYesNo `( Text : String):Boolean;`

Play a sound file.

Procedure Sound `( SoundFile : String);`

```
Ex : Sound(C:\WINDOWS\Media\notify.wav);
```

Exchange high order byte with the low order byte of word

Procedure XChg16 `( Var V : Integer);`

```
Var
  V : Word; ou V : Integer;
V:=$FF11;
Xchg16(V);  //** V is now equal to $11FF
```

Convert null-terminated string to a Pascal string

Function StrPas `( Var PStr ):String;` The PStr is a pointer on Characters (C language type).

These two functions compare two zones of bytes starting with the address included in B1 and B2.
The comparison is performed on consecutive bytes pointed by the NByte variable. CompareBlock will return true and CompareBlockEX will return 0 if the two zones are equivalent, otherwise the different address is returned.

Function CompareBlock `( Var B1,B2;  Nbyte : Integer) : Boolean;`

Function CompareBlockEx `( Var B1,B2 ; Nbyte : Integer)  : Integer;`

The GetTick function retrieves the number of milliseconds that have elapsed since Windows was started.

Function GetTick `: Integer;`

StrPCopy copies a Pascal-type string Source into a null-terminated string Dest. It returns a pointer to Dest

Function Pchar `( Var Str : String) : Integer;`

Function Random `( Range : Integer) : Integer;`

Function RandomFloat `( Range: integer)  : Real;`

Generates random numbers within a specified range. The result can be an integer or a real.
Random returns a random number within the range 0 <= X < Range.
Random is for an integer returned value.
RandomFloat is for a real returned value.

Function DialogBox `( Caption, Texte : String; Utype : Integer) : Integer;`

MessageBox is an encapsulation of the Windows API MessageBox function.
Utype is the dialog type box.

<u>These values can be combined to obtain the desired effect.</u>

```
MB_OK  MB_OKCANCEL  MB_ABORTRETRYIGNORE  MB_YESNOCANCEL MB_YESNO MB_RETRYCANCEL
//******* Combinaison of icons  *******
MB_ICONHAND, MB_ICONQUESTION, MB_ICONEXCLAMATION, MB_ICONASTERISK, MB_USERICON,
```

It returns one of the following values:

```
IDABORT or IDCANCEL or IDIGNORE or IDNO or IDOK or IDRETRY or IDYES
```

Windows Library

This library is based on the windows API (Procedure and function available in the Windows system DLL). We enter directly the procedures and functions of the windows API (See Microsoft SDK).

.

Function Open_File `(Var NomFichier : String; TypeFichier, Filtre, Titre : String) : Boolean;`

Function Save_File `(Var NomFichier : String; TypeFichier, Filtre, Titre : String) : Boolean;`

Is the standard action for displaying a file opened in the form of dialog
Is the standard action for displaying a file saved in the form of dialog.

.

Procedure Set_Focus `(Control : String);`

Change focus for a component. This component is Windows control type Listbox, Combobox, Checkbox, Radiobutton, grid, Editbox. The focus is given when the cusor allows editing.

Return true value if the form has the focus (Title bar active).

Function GetWinFocus `: Boolean;`

Refresh and repaint all the components present in the form.

Procedure FormRefresh;

Function ShellExecute `( hWnd: Integer; Operation, FileName, Parameters, Directory : PAnsiChar; ShowCmd: Integer): HINST;`

The ShellExecute function opens or prints a specified file. The file can be an executable file or a document file
Operation specifies the operation to perform.

'open' The function opens the file specified by FileName.
The file can be an executable file or a document file. The file can be a folder to open.

'print' The function prints the file specified by FileName. The file should be a document file. If the file is an executable file, the function will open the file, as if "open" had been specified.

'explore' The function explores the folder specified by FileName.

ShowCmd See constants in Windows.pas (SW_HIDE, SW_MAXIMIZE, SW_MINIMIZE Etc..) specifies how the application is shown when it is opened.

Example below
To open Excel normally with the Histo.dbf file. The interest of this instruction that you cannot know the complete path of Excel

```
ShellExecute(0, PChar('Open'), PChar('excel.exe'), Pchar('Histo.dbf'), nil,
SW_SHOWNORMAL);
```

To explore the C:\Windows directory you can use this call :

```
ShellExecute(0, Pchar('explore'), Pchar('C:\WINDOWS'), 0, 0, SW_SHOWNORMAL);
```

For the others functions and procedures, refer to Windows API.

MultiTasking and Threads

A *thread* is the smallest unit of execution within a process. Each thread has its own stack and registers; other resources of a process, such as global data, files, pipes, etc., are shared by all of its threads. Any process has at least one primary thread, which is also called the main thread or thread 1. It is started by the operating system when the process is created and creates other threads when necessary. When the main thread terminates, the whole process will terminate.
All threads of a process are executed concurrently in the sense that the operating system allocates small amounts of CPU time (*time slices*) to threads according to their *priorities*. This process is called *scheduling.*

Using thread can improve your application by managing your communication I/O cards while distinguished from the tasks in different priorities. For example, a thread of high priority manages critical tasks whereas a thread of low priority manages the other tasks.

The use of the threads must be done by respecting the following recommendations:
Preserving a too great number of threads wastes CPU time on a system, which has only one processor. The limit is of 16 threads per process.
If several threads update the same resources, you must synchronize the threads to avoid the conflicts

Function Thread (Proc : Integer; Priority : Integer) : Integer;
Function DestroyThread (hThread : THandle) : Boolean;
Procedure Sleep (dwMilliseconds : Integer);
Function ResumeThread (hThread : THandle) : Integer;
Function SuspendThread (hThread : THandle): Integer;
Function SetThreadPriority(hThread: THandle; nPriority: Integer): Boolean;

Example
To create 2 independent and parallel tasks in the main program

```
//*** Task 1 ***************
Procedure Task1;
Begin
  Repeat
    A0:=A0+1;
    Sleep(10); //** Give the hand to the OS
  Until False;
End;
//*** Task 2 ***************
Procedure Task2;
Begin
  Repeat
    A1:=(A1*2)+10;
    Sleep(20);  //** Give the hand to the OS
  Until False;
End;

{*** Button1 ***********************}
SObject Procedure XPBUTTON1;
Var
  ID1,ID2 : Integer;
Begin
 If Start_visu Then
 Begin
   ID1:=Thread (Addr(Task1),6); //** Start Task 1
   ID2:=Thread (Addr(Task2),6); //** Start Task 2
 End;
 If Button then
 Begin
   Destroythread(ID1); //** Destroy Task 1
   Destroythread(ID2); //** Destroy Task 2
 end;
End;
```

Function Thread (Proc : Integer; Priority : Integer) : Integer;
Proc : Procedure address executed by the thread.
Priority : Determine the thread's scheduling priority relative to other threads in the process

0 The thread is executed only when the system is idle.
The system will not interrupt other threads to execute a thread with tpIdle priority.
1 The thread's priority is two points below normal.
2 The thread's priority is one point below normal.
3 The thread has normal priority.
4 The thread's priority is one point above normal
5 The thread's priority is two points above normal.
6 The thread gets highest priority.

Retour : Thread Handle created.

Function DestroyThread (hThread : THandle) : Boolean;
Destroys the thread object and releases the memory allocated to it

Procedure Sleep (dwMilliseconds : Integer);
The Sleep function suspends the execution of the current thread for a specified interval.
The Task1 procedure is infinitive loop which does not block the system because it is a thread.

```
Procedure Task1;
Begin
  Repeat
    A0:=A0+1;
    Sleep(10); //** relinquish during 10ms
  Until False;
End;
```

Function ResumeThread (hThread : THandle) : Integer;
Function SuspendThread (hThread : THandle): Integer;

ResumeThread restarts the execution of a suspended thread
SuspendThread Pauses a running thread.
Call Suspend to temporarily stop execution of the thread. To resume execution after a call to Suspend, call Resume. Calls to Suspend can be nested; Resume must be called for the same number of times as Suspend was called before the thread will resume execution.

Function SetThreadPriority (hThread: THandle; nPriority: Integer): Boolean;
Determines the thread's scheduling priority relative to other threads in the process. It is not necessary because the priority is already set in Thread procedure.

Attention: "To inflate" the priority of the thread for an operation. Using the CPU intensively can "underfeed" the others threads of the application. One should give a high priority only to threads that pass the main part of time to await external events.

Execution of a program on keyboard or mouse events

The **EventAPP** variable indicates whether a keyboard or mouse event performed in the form. Thanks to this variable, you can execute actions or treatments on clicking mouse or keyboard events made by the operator.

Example :

```
{*** Value1 ****************************************}
SObject Procedure Value1;
Begin
  If EventAPP Then //** Calculation only done on keyboard or mouse event
  Begin
    R1:= Exp(R2) / 2;
    R1:= R1 + R2 / R1;
  End;
End;
```

Enables or disable closing the form

The **SetCanclose** procedure enables closing when the user attempts to close the form.

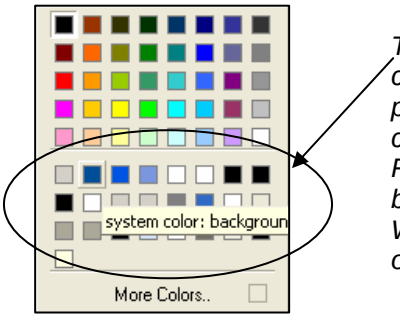
```
{*** Button1 ****************************************}
SObject Procedure XPBUTTON1;
Begin
 SetCanClose(False); //** Disable closing the form
 If Windisparition Then Message('Ciao'); //** Closing the form
 If Button then CLoseAppli; //** Quit the form programmatically
End;
```

System Colors

To indicate a color exists in Visual I/O, the TColor type is an integer. Several types such as Tcolor are integers (THandle, Dword etc.). A TColor variable represents a color:
If you specify TColor as a specific 4-byte hexadecimal number instead of using the constants defined in the Windows unit, the low three bytes represent RGB color intensities for blue, green, and red, respectively. The value $00FF0000 represents full-intensity, pure blue, $0000FF00 is pure green, and $000000FF is pure red. $00000000 is black and $00FFFFFF is white..

The Windows unit contains definitions of useful constants for TColor. These constants map either directly to the closest matching color in the system palette (for example, clBlue maps to blue) or to the corresponding system screen element color defined in the Color section of the Windows Control panel (for example, clBtnFace maps to the system color for button faces).

If the highest-order byte is zero ($00), the color obtained is the closest matching color in the system palette. If the highest-order byte is one ($01), the color obtained is the closest matching color in the currently realized palette. If the highest-order byte is two ($02), the value is matched with the nearest color in the logical palette of the current device context.



The palette of color of the editor enables choosing one of the system colors in Windows. To choose one of the system colors makes it possible to follow the definition of those when you change topic in the control panel of Windows.
For example, the color Button Face which regulates the color of a button is defined gray but with nuances of gray between traditional Windows and Windows XP. While choosing this system colo,r the color of your buttons will follow the change of theme automatically.

Function ColorToRGB(Color: TColor): Integer; *//** Convert a TColor value into an RGB representation of the color*
Function GetSysColor(nIndex: Integer): TColor; *// Retrieve the current color of the specified displayed element*

The INI files

The **INIFILES** VPU stores and retrieves application-specific information and settings from INI files.

It enables handling the storage and retrieval of application-specific information and settings in a standard INI file. The INI file text format is standard introduced in Windows 3.x for storing and retrieving application settings from session to session. An INI file stores information in logical groupings, called “sections.” For example, the WIN.INI file contains a section called “[Desktop]”. Within each section, actual data values are stored in named keys. Keys take the form:

<keyname>=<value>

A FileName passed to each function is the INI file name.

Functions list

Function IniFileWriteString (FFileName,Section, Ident, Value: string) : Boolean;
Function IniFileWriteBool (FileName, Section, Ident: string; Value: Boolean): Boolean;
Function IniFileWriteInteger (FileName, Section, Ident: string; Value: Longint): Boolean;
Function IniFileWriteTime (FileName, Section, XName : string; Value: TDateTime): Boolean;
Function IniFileWriteDate (FileName, Section, XName: string; Value: TDateTime): Boolean;
Function IniFileWriteDateTime(FileName, Section, XName: string; Value: TDateTime): Boolean;
Function IniFileWriteFloat (FileName, Section, XName: string; Value: Real): Boolean;
Function IniFileReadString (FFileName, Section, Ident, Default: string): string;
Function IniFileReadBool (FFileName, Section, Ident: string; Default: Boolean): Boolean;
Function IniFileReadInteger (FFileName, Section, Ident: string; Default: Longint): Longint;
Function IniFileReadTime (FFileName, Section, XName: string; Default: TDateTime): TDateTime;
Function IniFileReadDate (FFileName, Section, XName: string; Default: TDateTime): TDateTime;
Function IniFileReadDateTime(FFileName, Section, XName: string; Default: TDateTime): TDateTime;
Function IniFileReadFloat (FFileName, Section, XName: string; Default: Real): Real;

Writing example in a INI file (Fungame.ini).

```
If Button Then
  Begin
     IniFileWriteString ('FUNGAME.INI','Software', 'Name 1', 'Visual I/O');
     IniFileWriteString ('FUNGAME.INI','Software', 'Name 2', 'Visual PLC');
     IniFileWriteBool   ('FUNGAME.INI','Options', 'Sound', True);
     IniFileWriteInteger('FUNGAME.INI','Options', 'Level', 3);
     IniFileWriteBool   ('FUNGAME.INI','Configuration', 'ShowToolBar', True);
  End;
```

Result in Fungame.ini file :

[Software]
Name 1=Visual I/O
Name 2=Visual PLC
[Options]
Level=3
Sound=1
[Configuration]
ShowToolBar=1

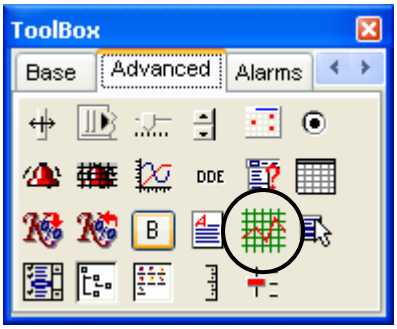
Case of reading

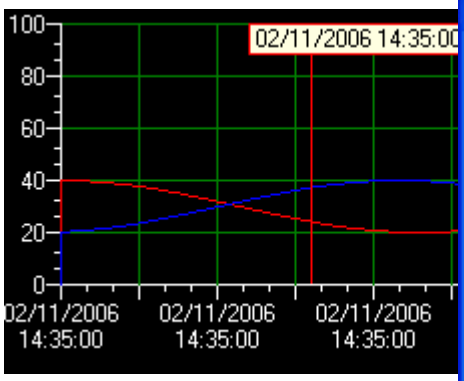
```
    TS:= IniFileReadString ('FUNGAME.INI','Software', 'Name 1','ARSOFT Visual');
```

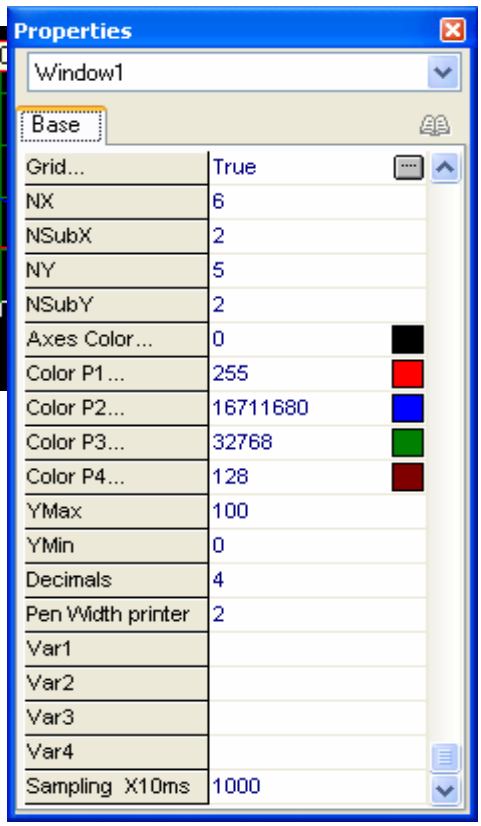
In that case the value returned in TS is Visual I/O. However, if the Software section does not exist or the keyname (Name 1) does not exist, a default text value is returned (here ' ARSOFT Visual').

IniFileReadString reads a string value from an INI file. Section identifies the section in the file that contains the desired key. Ident is the name of the key from which you can retrieve the value. Default is the string value to be returned if the:
Section does not exist. Key does not exist. Data value for the key is not assigned.

Trend Histo - The experimental component







Grid Show a grid. The grid color is given by the color contour **Color P1, P2, P3, P4** Pen color of the trend lines.
NX,NSubX,NY,NSubY : Number of minor and major ticks in X & Y axes.
YMax et Ymin Minimum and maximum Y axes.
Decimals : Precision Y axes of the text
Pen width Printer. Pen width in case of printing of the curves.
Var1 to Var4 : (optional) Variables names to be recorded and displayed in real time..
Sampling : Value to multiply per x10ms for the schedule rate for displaying and recording news values (Var1 to Var4).

This component allows viewing trend graph where 4 maximum variables are recorded at a defined schedule rate (Sampling) . The trend viewer is persistent, that is, when you start the system or change displays, the historical data on the display is retained. The component needs additional components to operate as you want. These additional components are placed around the trend viewer and can be grouped if several trend viewer are displayed on the form. Otherwise, if you use only one trend viewer on a form, it is not necessary to group the additional components. Trend viewer ,along with the additional components, creates a complex or simple trend viewer finally.
An internal circular buffer for the 4 trend lines (Points X = Date Y=Valeur) of 21600 records are available.
If only one variable is set (record at a sampling time), this value of this variable is recorded into the internal buffer in the 21600 last record.
If 2 variables are set, then 21600 / 2 = 10800 records will be affected to these 2 variables, if 4 variables (Var1 to Var4) are set, each buffer will be 21600 /4 = 5400 records maximum..

VPU SHIFTTREND.PAS
Library allows the interface with this component. The procedures and functions are based on sending Windows messages to the trend viewer. Before working with this type of component, it is necessary to recover its Handle by the **SHTGetTrendHandle** function, for example to add additional points to the trend viewer.
All is based on passing parameters through a huge record.
Example of displaying to hide the grid:

```
//**** Grid/No Grid **********************
Procedure SHTGridVisible( On : Boolean);
Begin
  POneTrendAttrib:=SHTGetAttributsADDR;          (1)
  If POneTrendAttrib=0 then Exit;
  TrendAttributs(POneTrendAttrib^).Grid:=On;     (2)
  SHTUpdateTrend;                                (3)
End;
```

To send a new value, it is necessary to :
(1) Get the component handle. If this one is equal to 0, then ERROR.
(2) Typecast this pointer to modify the record element. Here we modify the grid flag.
(3) Ask the component to refresh its drawing

Detail of the internal record to communicate with the trend viewer.

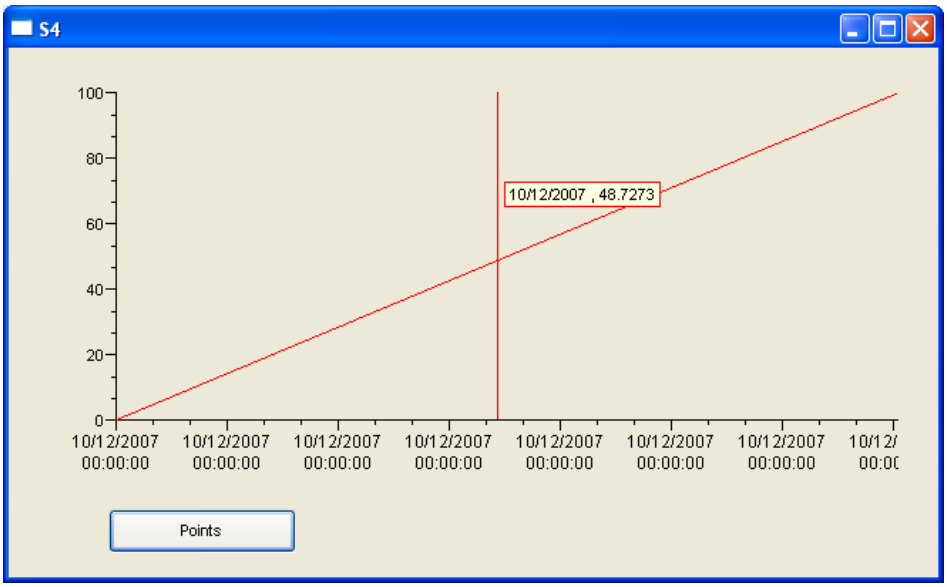
```
TrendAttributs = Record
  CustomTrace     : Boolean;  //*** Drawing custom or normal
  PenWidth        : Integer;  //*** Pencil Width
  PrintPenWidth   : Integer;  //*** Printer Pencil Width
  Grid            : Boolean;  //*** grid shown
  NBDivY          : Integer;  //*** Tip
  NBDivX          : Integer;
  NBSubX          : Integer;
  NBSubY          : Integer;
  YHigh           : Extended; //*** Maximum Axe Y
  YLow            : Extended; //*** Mini Axe Y
  Decimal         : Integer;  //*** precision (#decimals) on Y axis
  Step            : Integer;  //*** Step between 2 points
  ColorAxis       : TColor;    //** Color for the 2 axes
  ColorTextAxis   : TColor;    //** Text color on the Axes X & Y
  FontHeightX     : Integer;   //** Text height on X axis
  NBCourbes       : Integer;  //*** de 1 to 4 maxi
  ColPens         : Array [1..MaxCourbes] of TColor; //** Trend line Colors
  Scales          : Array [1..MaxCourbes] of Extended; //*** Coef for each point
                                          (original coefficient =1)
  ListPoints      : Array [1..MaxCourbes] of Pointer; //*** List of points
  ListPointsHist  : Array [1..MaxCourbes] of Pointer; //*** List of histo points
   RecordVars      : Array [1..MaxCourbes] of String; //*** Varaibles names to
                        record cyclically (Var1 to Var4 in properties box)
  FirstPtVisible : Integer; //*** N of the first point visible at left
  BarChannel      : Integer; //*** # of trend line associated to the mobil
                              vertical cursor
  XChannel        : Integer; //*** Which trend line Graduates the X Axis
  HintVisible     : Boolean; //*** Vertical Bar(cursor) visible
  XYPlot          : Boolean; //*** Displayed in DateTime or Point on X axis
  inHisto         : Boolean; //*** In historisation or au fil de l'eau
  SampleTime      : Integer; //*** Schedule rate in X10 ms to display and record
                             new points
  StopRecord      : Boolean; //*** Stop or run display
  GotoEnd         : Boolean; //*** Goto the last recorded point and enables the
                             trend to shift when a new point is recorded
  //***** Informations retruns by the system read only mode ***************
  RetFrameWidth   : Integer;  //*** Return the width of the drawing frame
  RetAdrPosi      : Integer;  //*** Pointer – Display Position in % on the fifo
  RetNewPos       : Integer;  //*** Pointer - Position on the fifo requested
  RetPosCursor    : Integer;  //*** Position vertical Cursor
  RetPtCursor     : Integer;  //*** Pointer on the Couple Date/Value
  RetPtPoint      : Integer;  //*** Pointer on the Couple Date/Valeur first
                             point drawn
  RetMaxPoint     : Integer;  //*** maximum number of points recorded by canal
  RetMaxiY        : Extended; //***  Maxi Y Value
  RetMiniY        : Extended; //***  Mini Y Value
End;
```

RetAdrPosi : If <>0 is a pointer on an integer variable that is refreshed automatically by the trend viewer. This variable indicates the position (in %) of the first point drawn in the fifo (the most left in the component).
RetNewPos : If <>0 is a pointer watched by the trend viewer indicating in the case of changing the new position (in %) in the current fifo for starting its display from left to right.

For adding new functionalities, you can report to the provided source SHIFTTREND.PAS. This component can be evolved with future version of Visual I/O.

Add points to a trend by program.

This example allows to draw by program points (linear curve) in a trend viewer without using the components of trends tab. An automatic detection of the useful area allows to add the number of points consequently. Place trend viewer on your form.



The the program behind the button

```
{** Variables & Procedures **}
Uses ShiftTrend;  //*** Use the library to control the trend
{*** Window1 ***************************************}
SObject Procedure Window1;
Begin
End;
{*** XPBUTTON1 ***************************************}
SObject Procedure XPBUTTON1;
Var
  HDle,I,L        : Integer;
  POneTrendAttrib : Integer;
  Pas,V,R         : Real;
Begin
  If Button Then
  Begin
    GO2('Window1');
    HDle:=SHTGetTrendHandle ; //** Get the Trend Handle
    SHTClearAllPoints(255);   //** Clear all trends
    L:=SHTGetWidth;           //** Get the Number of Points in X
    R:=L;
    Pas:=100 / R;  //*** internal Step
    V:=0;
    For I:=1 to L do
    Begin
      SHTAddpoint (HDle,1,CDate,V ,0,False);  //*** Add a point
      V:=V+Pas;                               //*** Inc value
    End;
    SHTUpdateTrend; //*** finally refresh the trend
  End;
End;
```

Wide strings

Visual I/O uses the String type, which is a short string (255 characters maximum) with the first byte given [0] the string length. However, it is necessary to arrange character strings of big lengthes (> 255 car) in certain cases. The PChar Type is a pointer on characters, which is possible by using the Getmem procedure that allocates a number of bytes in the memory. The difficulty is to manipulate these PChar as normal strings.

Null terminated string is a sequence of ASCII characters, one to a byte, followed by a zero byte (null). Null terminated strings are common in C and C++.

The StrToPchar function transforms a classical string into pointer on characters (Pchar).
For example:

```
Var
  P : Integer;
Begin
  P:=StrToPChar('Visual I/O');
```

The StrAdd function adds a string to a Pchar.

```
P:=StrAdd(P,' Development Tool');  //**P pointer on 'VisualI/O Development Tool'
```

StrLen Returns the length of the PChar string.
StrEnd Returns the address of the last character in the PChar.
StrComp Compares the two Pchar and returns 0 if they are equal.
The returned value <0 if Str1< Str2, =0 if Str1= Str2, >0 if Str1 > Str2
StrMove Copy exactly Count characters from Source to Dest and returns Dest. Source and Dest can be covered.
StrAlloc Allocates a buffer for a Null terminated string and returns a pointer on the first character.
StrBufSize Returns the size of the buffer allocated by strAlloc.
StrNew Allocates a copy of STR in the memory.
StrDispose Releases a string allocate by StrAlloc or StrNew
StrConCat Concats Destination + source and returns the pointer on the first character.
StrCopy Copys Source to Dest and returns Dest

List of the routines to manage the wide strings

```
//****** Pchar procedures ****************************
 Function  StrLen     ( Str : Pointer ): Integer;
 Function  StrEnd     ( Str : Pointer ): Pointer;
 Function  StrComp    ( Str1, Str2 : Pointer): Integer;
 Function  StrMove    ( Dest, Source : Pointer; Count: Integer):Pointer; //PChar;
 Function  StrAlloc   ( Size: Integer): Pointer; // PChar;
 Function  StrBufSize ( Str : Pointer ): Integer;
 Function  StrNew     ( Str : Pointer): Pointer;
 Procedure StrDispose ( Str : Pointer);
 function  StrConCat  ( Dest, Source: Pointer): Pointer; //PChar;
 Function  StrAdd     ( CurrentPChar : Pointer; Source : String) : Pointer;
 function  StrCopy    ( Dest, Source: Pointer): Pointer;
 Function  StrToPChar ( S : String) : Pointer;
```

The WinSock library

Winsock provides a set of functions, data structures...etc required to access the network services of any protocol stacks. Winsock acts as a link between network applications and underlying protocol stacks. These procedures let you create an application that can communicate with other systems using TCP/IP and related protocols. Using sockets, you can read and write over connections to other machines without worrying about the details of the underlying networking software. Attention: this library is used by Openmodbus, if you want to modify this library. It is preferable to create a new library (for example, WinsockEx.pas). For non-blocking applications, it is recommended to create an independent thread for managing the socket communications. You can see the PCOpenModbus.pas generated by the configurator (openmodbus settings in the main menu).

Procedures and functions

la combination **address IP + port** is an unique address, it is calling **Socket**

```
//*** Open a socket
Function SocketOpenConnection (IPA : String; Port : Integer;
                                    Var DSocket : Integer) : Boolean;
IPA : IP Address;       Port: communication Port;
Return : Return True if the socket is open.
//*** Close the socket.
Procedure SocketCloseConnection ( Var DSocket : Integer);
//*** Wait for a number of bytes coming from the socket or 500ms of time out.
Function SocketWait (Var SocketHandle : Integer; NB : Integer) : Boolean;
//*** read the reception buffer of the socket.
Function SocketReceiveBuf (Var SocketHandle : Integer;
                                var Buf; Count: Integer): Integer;
//*** Send bytes into a socket
Function SocketSendBuf(Var SocketHandle : Integer; var Buf; Count: Integer):
Integer;
//*** Return the number of byte containing in the reception buffer of the socket
Function SocketReceiveLength (Var SocketHandle : Integer) : Integer;
//*** Get host information corresponding to the pc IP address
Function GetHost_IP     : String;  //*** Example "192.168.0.1"
//***  Get host information corresponding to a hostname
Function GetHost_Name   : String;  //*** Example "ShoeBox"
```

Example writing and reading in a socket.

```
Var
   MySocket : Integer;
   Obuf     : Array [0..30]  Of Byte;
   IBuf     : Array [0..500] Of Byte;
   I        : Integer;
Begin
  If  SocketOpenConnection ('192.168.0.100',80, MySocket) Then
  Begin
     FillChar(Obuf,Sizeof(OBuf),0); //** Reset local emission buffer
     Obuf[5] :=3; Obuf[6] :=12;
     Obuf[7] :=7; Obuf[8] :=70;
     I:=SocketSendBuf(DSocket,OBuf,9); //** Send 9 bytes
     If I<>-1 Then     //** If the sending is ok, then I<>-1
     Begin
      If SocketWait(DSocket,8) Then //** Wait for 8 Bytes minimum in return
      Begin
        I:=SocketReceiveBuf(DSocket,IBuf, 500); //** Get All the buffer
                                                // maximum 500 bytes
        If I>=8 then //I contains the number of bytes read.
        Begin
          Move(Ibuf[0],TabloRetour[0],I); //** Treat the buffer
        End;
     End;
     SocketCloseConnection(MySocket); //** Close the socket
  End;
End;
```

Multi-monitor applications

Visual I/O enables writing multi-monitor applications.
You can dispatch your forms over several monitors.
When a form is displayed on a specific monitor, the other forms launched from this form will be also displayed on the same monitor.
That enables managing navigation on the same screen (Monitor).

Let us consider a system with three monitors. You must display first synoptic that is similar on each monitor.

You will develop form only once and you will use the procedure of duplication **DuplicateForm**.

For example:

```
If Button Then
Begin
  Display('MainMenu'); //** Display on the primary monitor
  Duplicateform ('MainMenu',2,2);        //** The same on Monitor #2
  Duplicateform ('MainMenu',3,3,20,550); //* The same on Monitor #3 X=20 & Y=550
End;
```

Procedure DuplicateForm (**FormName** : String; **Indice** : Integer; **Monitor** : Integer; **NewPosX, NewPosY** : Integer);

FormName : Form name to duplicate.
Indice : Indice number to be added to the end of the form name duplicated.
Monitor : Monitor number. The primary monitor is Number 1.
NewPosX : New X position on the monitor (Optional).
NewPosY : New Y position on the monitor (Optional).

A new form is created in the application, this one is rigorously identical to the original with the name FormName+Indice

Example `Duplicateform ('MainMenu',2,2);` The name of form that will be displayed on the monitor number 2 is MainMenu2.

In the example above, MainMenu is duplicated in MainMenu2 and MainMenu3. It can individually call the same forms. If Mainmenu displays a form, this one is displayed on the same monitor. If a mainmenu of another monitor calls the same form, this one is moved from the previous monitor to a new monitor. In other words, the form cannot be displayed on two monitors at the same time.

If necessary, it will also be necessary to duplicate the forms to display in multi instance.

Procedure **DisplayEX**(FormName : String; Monitor : Integer);

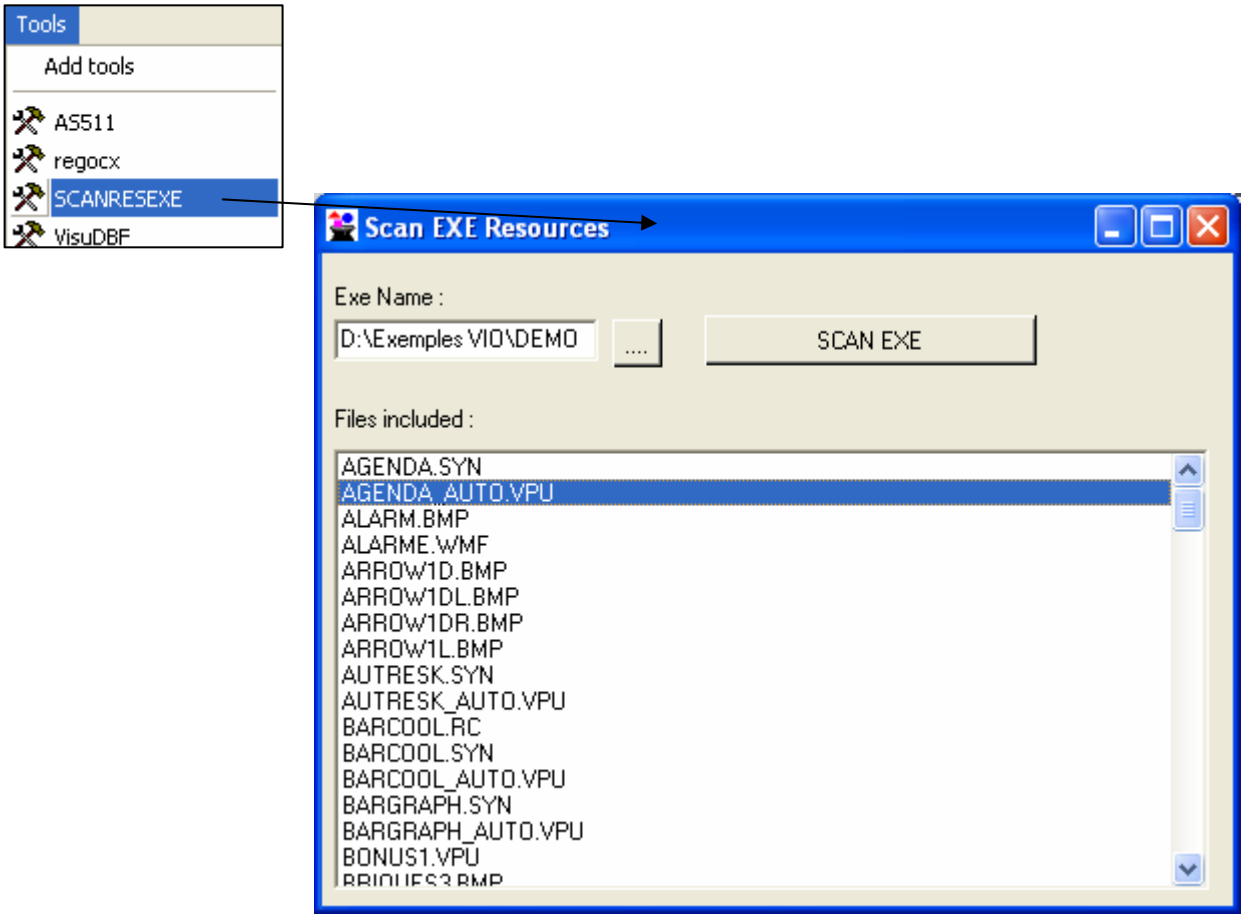
This procedure is an extension of the display procedure. It enables displaying a form on a specific monitor. If the monitor does not exist, the form will be displayed on the main monitor.
The parameter Monitor can be missing (optional). In this case, this procedure acts as displaying procedure.

For example:

```
DisplayEX ('SumAlarms',2);  //** Display the form SumAlarms on monitor #2
DisplayEX ('Trends'); //*** Display the form Trends on the primary monitor.
```

SCANRESEXE.EXE Utility

This program enables scanning the resources of an executable file created with Visual I/O
This utility enables controlling if all the specified files are correctly included in the final executable final, which will be transferred onto the final PC of your application
If this utility is not present in your PC, download it on ARSOFT web site by Upgrade menu



Choose the executable file by clicking on the button on the right of the edit box, then click on SCAN EXE.
The lower listbox displays all files included in the resource of the executable created by visual I/O

☞ **Note**: This utility allows only the visualization.