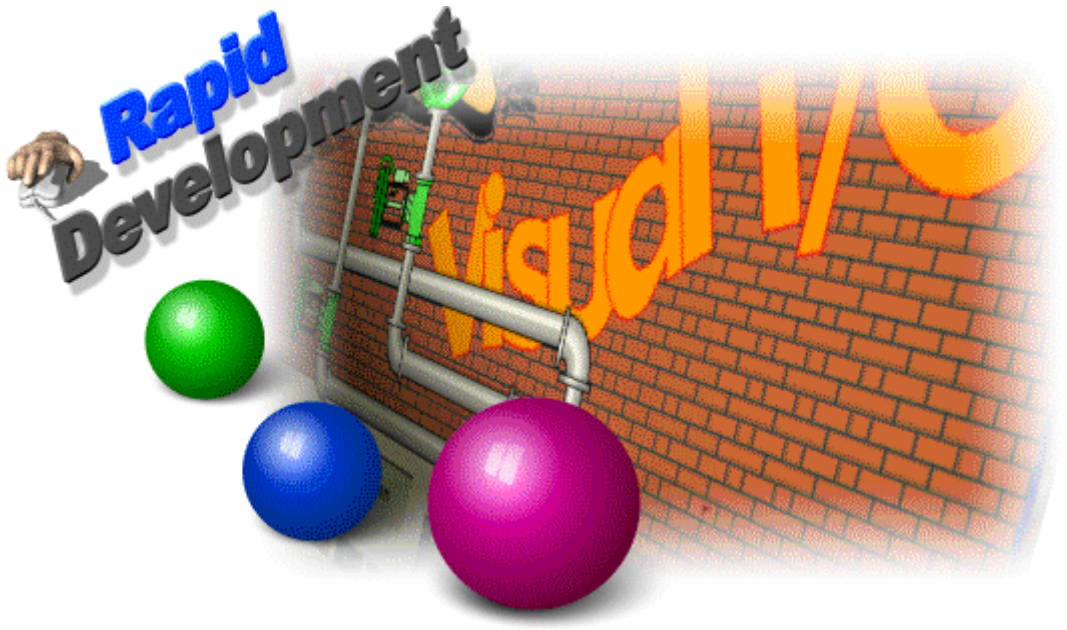
Visual I/O

Productivity tools

Manuel de Programmation



- **Editeur**
- **Experts**
- **Langage**
- **Programmation avancée**

Ver 7.xx

Philosophie Visual I/O

Visual I/O est un outil de développement générant des applications d'interfaces Homme/Machine s'exécutant sur micro ordinateur compatible PC et sous système d'exploitation Windows NT,2000 et XP.
Visual I/O intègre un moteur temps réel, pouvant exécuter en tache de fond des programmes écrits en Grafcet, Schéma à contacts ou langage évolué. Les programmes écrits en Grafcet et Ladder par **Visual PLC** l'outil d'automatisme complémentaire de Visual I/O.
Visual I/O peut être connecté à des automates programmables via les drivers de communication spécifiques ou par son interface OPC client (Voir Menu principal Paramétrage puis paramétrage OPC). Afin d'assurer un dialogue permanent, ces dialogues avec les automates programmables sont exécutés par le moteur temps réel.

Visual I/O se décompose en deux grandes parties :

Une partie paramétrage des objets graphiques (Cercles, Boutons, Courbes Etc.) à l'aide de boites de dialogues.
Une seconde partie en programmation pure, permettant de programmer des actions spécifiques ne pouvant êtres mises en place par de simple boites de dialogue.
Visual I/O intègre le compilateur ARSOFT International. Celui-ci rendant l'application générée extrêmement rapide vis à vis des autres superviseurs du marché.
Visual I/O n'a besoin d'aucun outils tiers (comme VBA) pour le développement ainsi que l'exécution de vos applications.
Visual I/O génère des **fichiers Exécutables** (.EXE) incluant ainsi tous les fichiers et simplifiant ainsi toutes les installations ou les réinstallations éventuelles du à des plantages de l'OS.

Le langage de programmation choisi est le langage Pascal. Celui-ci allie puissance et simplicité. Près de 1000 Fonctions intuitives sont disponibles en mode programmation. Ce langage pourra être utilisé pour créer des symboles évolués, que vous placerez dans la boite à outils de l'éditeur ou tout simplement des librairies de fonctions réutilisables dans d'autres applications.

Visual I/O permet la constitution d'une application comprenant un ou plusieurs écrans appelés Synoptiques. Chaque synoptique contient des éléments graphiques (Cercles, Boutons, Courbes.).

Le mode Expert :
En double-cliquant sur chaque élément graphique, une boite de paramétrage apparaît. Pour chaque type d'élément, Visual I/O répertorie les fonctionnalités possibles (ex : un bouton peut être : simple, à accrochage, valide ou non, associé à une touche etc.).

Le mode Programme :
Pour les cas non prévus en mode expert, l'utilisateur peut passer en mode programmation par sélection de l'élément graphique puis un clic droit de la souris en choisissant la rubrique programme dans le menu flottant. Un éditeur de programme à coloration syntaxique apparaît permettant d'écrire du code associé à l'objet graphique.
Ce code est ensuite Compilé automatiquement 'au vol' à l'aide du compilateur ARSOFT International intégré dans Visual I/O.
Ce compilateur est optimisé et aussi performant que les compilateurs les plus connus.

Les drivers vers les automates programmables :
Grâce aux différents paramétreurs, vous pourrez définir des dialogues sous différents protocoles (Modbus, MPI Etc.).
Ces drivers utilisent les variables globales définies au fil du temps dans l'éditeur Visual I/O.
Ces drivers sont exécutés aussi dans le moteur temps réel, permettant d'avoir une communication la plus fluide possible quels que soient les traitements effectués en avant plan. La communication est prioritaire sur les synoptiques et leurs programmes.

Le Projet final compilé

L'application que vous développez est livrable sous forme d'un exécutable (.EXE) intégrant tous les fichiers nécessaires. Les fichiers supplémentaires nécessaires peuvent êtres aussi ajoutés manuellement dans cet exécutable (Menu utilitaires/Créer un exécutable).
Quand vous lancez l'application depuis l'éditeur, celui-ci le compile. Si la clef (dongle USB) est enfichée sur le PC, l'exécutable créé est libre de runtime et dupliquable à souhait. Si la clef USB n'est pas présent sur le PC, l'exécutable créé est en mode démo. Ce mode démo, affiche au lancement une boite de dialogue signalant ce mode puis s'arrête au bout de 15 minutes.

Je n'ai pas le temps de lire la doc

1 - Créer un nouveau projet

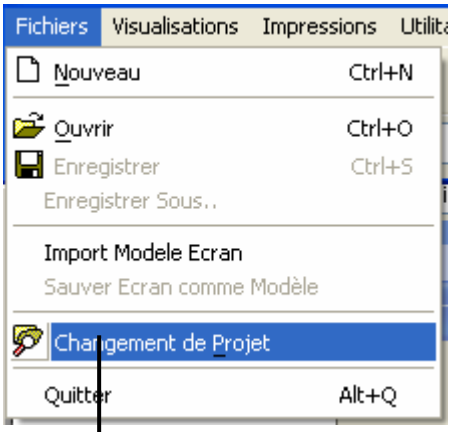
Sous Visual I/O, les projets sont désignés par des noms et des répertoires associés contenant tous les fichiers de votre application. Toutes ces informations sont stockés dans la base de registres de Windows © .

La sauvegarde complète de votre application est facilitée en copiant l'ensemble du répertoire.

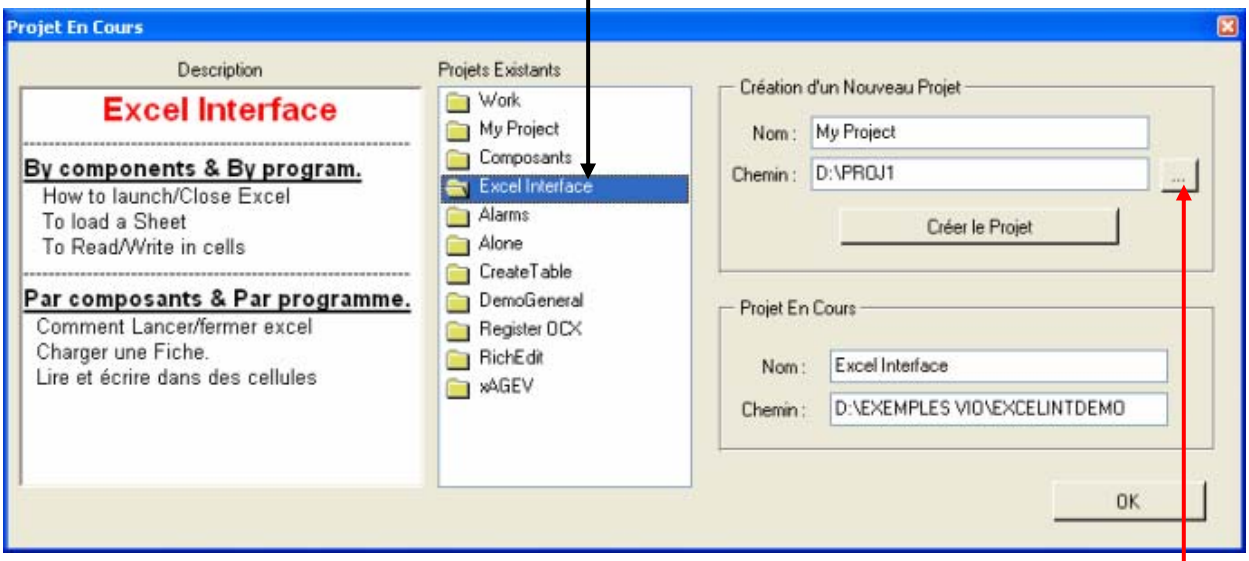
2 - Définir le nom du projet

Dans le menu *Fichiers*, cliquez sur *Changement de Projet* pour

- Changer de projet
- Créer un nouveau projet



Double cliquez sur le nom du projet afin de le désigner comme le projet courant

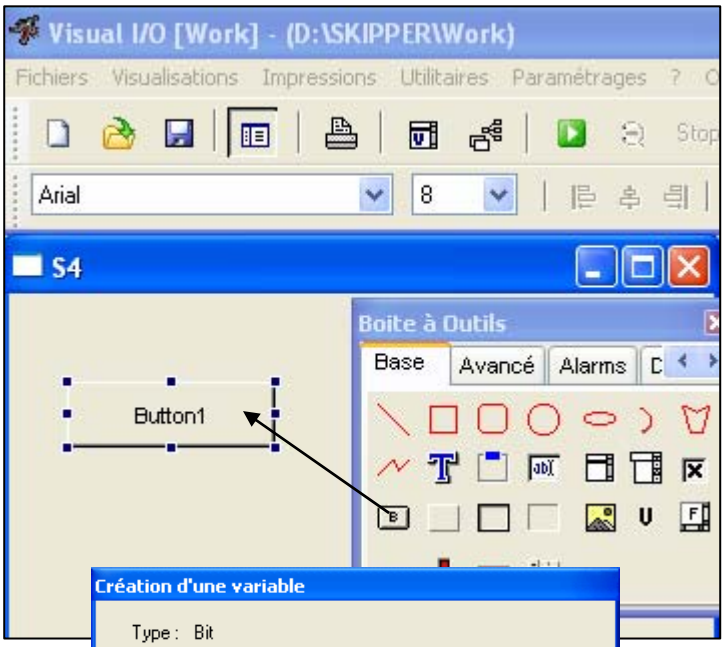


Indiquez dans le champ *Nom*, le nom de votre nouveau projet (ici My Project)
Indiquez dans le champ *Chemin*, le chemin complet ou doit se situer ce nouveau projet (ici D:\Proj1) par navigation
En cliquant sur le petit bouton. Si le répertoire n'existe pas vous avez la possibilité de le créer vous avez la possibilité de le créer.
Cliquez alors sur le bouton **Créer le Projet** afin d'enregistrer le nom et le chemin de celui-ci dans la base de registre de Windows. Ce projet passe alors dans la liste centrale des projets, il suffit de le sélectionner en double-cliquant dessus.

Note : Si vous créez un nouveau projet, aucune variable globale n'est définie. Vous devez les définir.

La description du projet (facultatif) est possible au format RTF dans la zone de texte à gauche. Pour obtenir les différentes fontes et effets graphique, constituez votre texte sous Word par exemple puis effectuer un copier/coller dans la zone de texte Description.

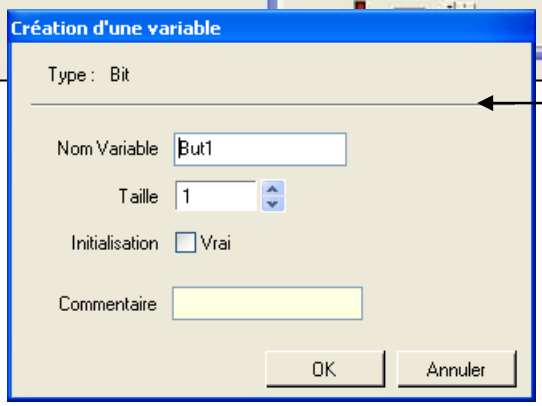
3 - Placer un bouton sur l'écran

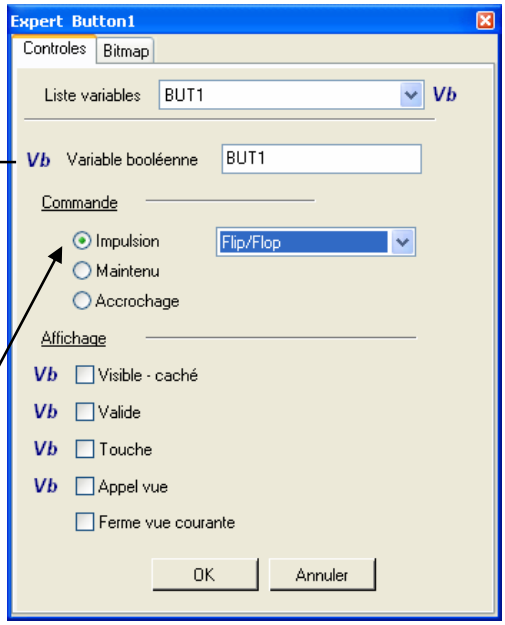


Appelez la boite à outils, choisissez un bouton puis placez le sur l'écran.

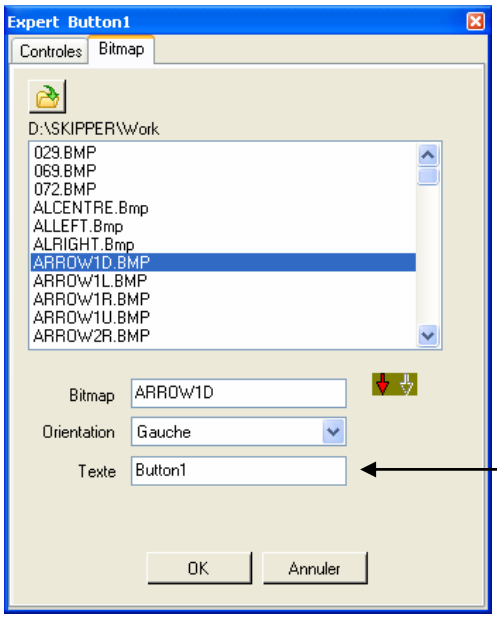
Double cliquez sur ce bouton pour faire apparaître la boite de paramétrages.

Frappez la variable que vous voulez affecter à ce bouton.
Ici **But1**





Ici le bouton est de type Flip/Flop, c'est à dire que le bit **But1** sera inversé à chaque clic de la souris.



Visible – caché : Bit permettant en fonction de son état d'afficher ou de cacher le bouton.
Valide : Bit permettant de valider/dévalider le bouton.
Touche : Touche du clavier associé.
Appel vue : Le bouton appel un autre synoptique.
Ferme vue courante : Ferme le synoptique.

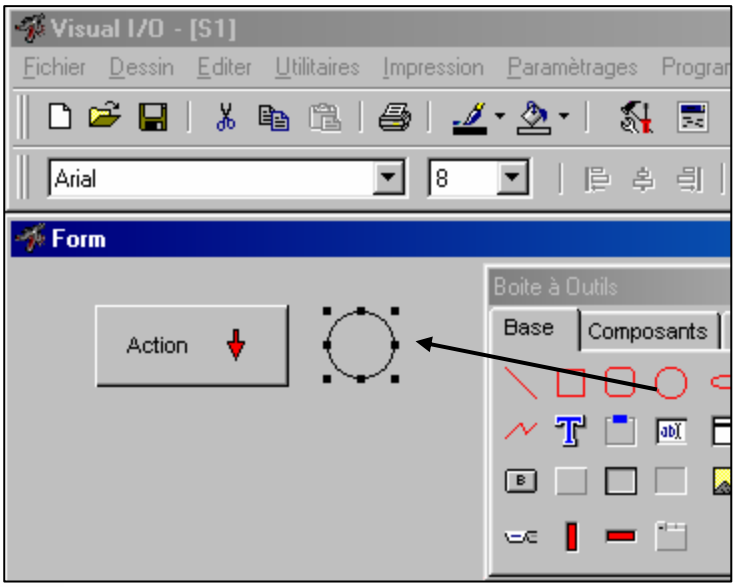
En final réglez le texte qui doit apparaître sur le bouton et éventuellement un petit Bitmap symbolisant son action

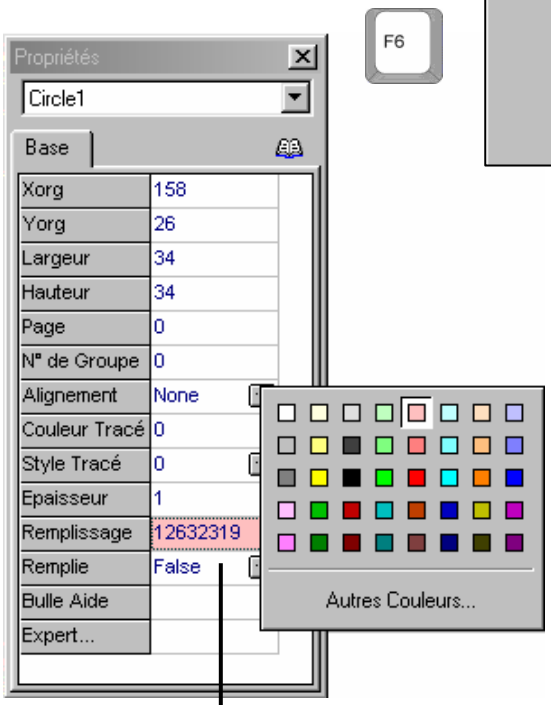
4 - Affichez l'état d'un bit (Booléen) à l'aide d'un cercle

Sélectionnez un cercle, un rectangle, une ellipse ou polygone pour la visualisation de l'état (par exemple le bit but1 forcé par le bouton).

Le cercle, rectangle, ellipse ou le polygone ont les mêmes possibilités d'affichage.

Cependant ceux-ci ne sont pas remplis d'origine.

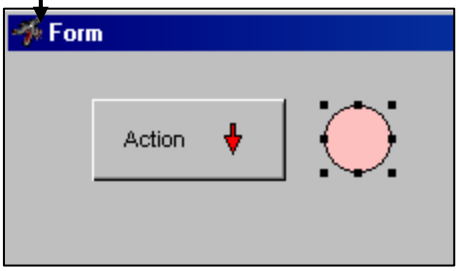


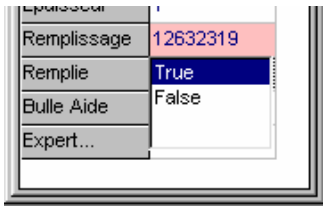


Pour choisir la couleur de remplissage par défaut (sans animations), Sélectionnez le cercle ou une autre figure, puis frappez la **touche <F6>** pour afficher ses **propriétés graphiques**.

Cette boite de propriétés renvoie une grille contenant les différents attributs graphiques réglables pour la figure sélectionnée. Ces attributs peuvent varier en fonction du type de la figure.

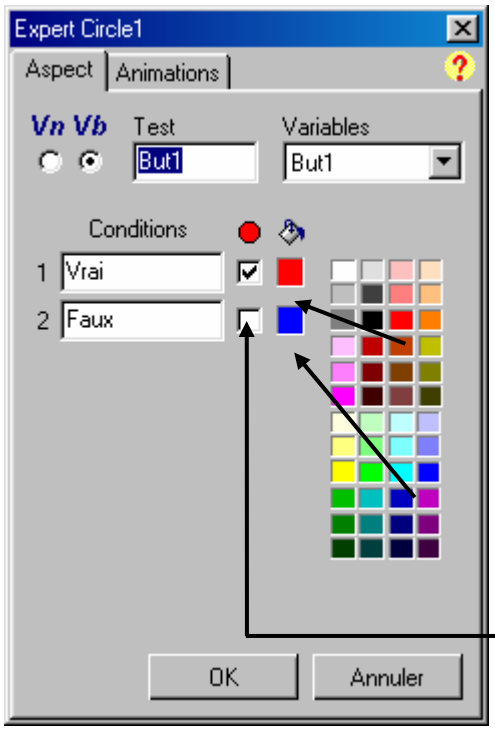
Pour choisir la couleur par défaut de cette figure, double-cliquez sur la cellule devant le texte remplissage.





5 - Paramétrage du cercle

Double cliquez sur celui-ci une boite de paramétrage apparaît :



Amenez la variable **But1** du ComboBox 'Variables' dans le champ de saisie Test par Drag & Drop ou frappez directement le nom de cette variable dans celui-ci.

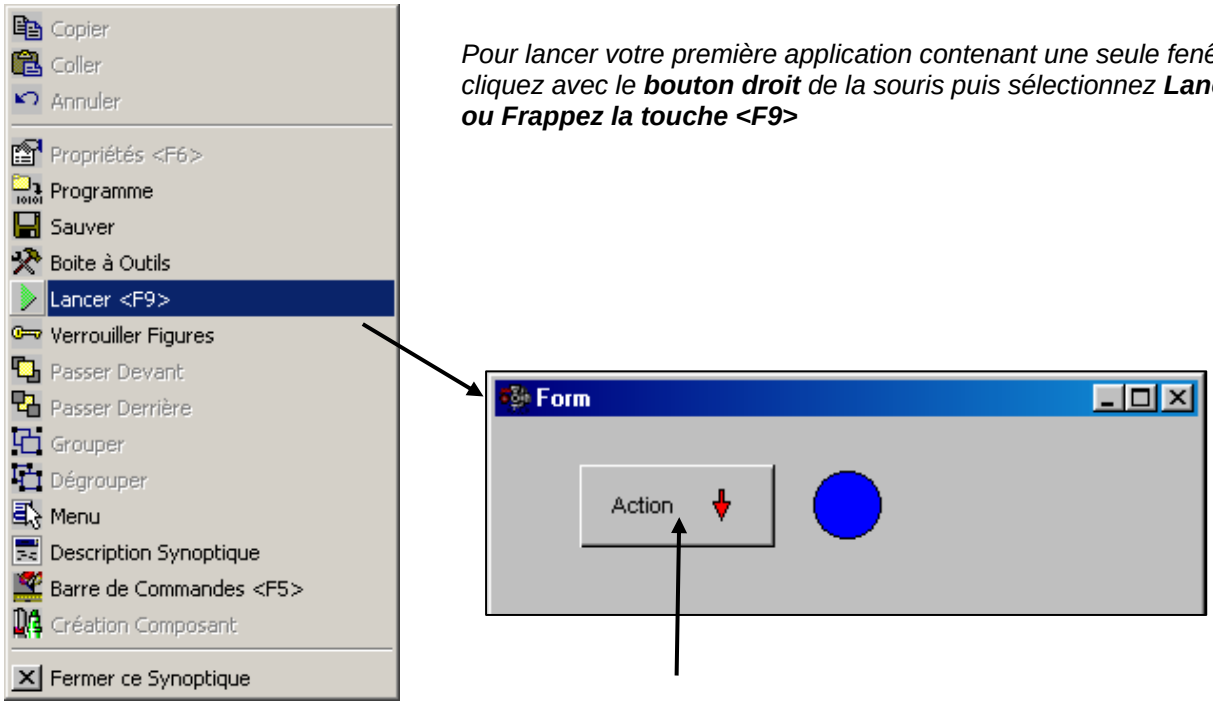
Drag & Drop : Cliquez à l'aide du bouton gauche de la souris sur le texte But1 sous le Combobox tout en gardant le bouton enfoncé, déplacez le curseur de la souris au-dessus du champ de saisie *Test* pour relâchez le bouton.

Pour l'affectation des couleurs, amenez la couleur désirée par drag & drop de la palette sur le carré.

Une couleur est affectée quand le bit est vrai mais aussi quand le bit est faux.

La case à cocher permet de faire clignoter la figure lorsque la condition est valide.

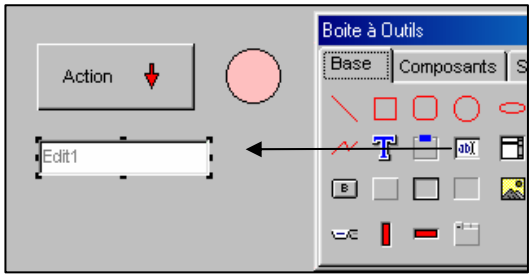
6 - Passez en mode exécution



*Pour lancer votre première application contenant une seule fenêtre, cliquez avec le **bouton droit** de la souris puis sélectionnez **Lancer ou Frappez la touche <F9>***

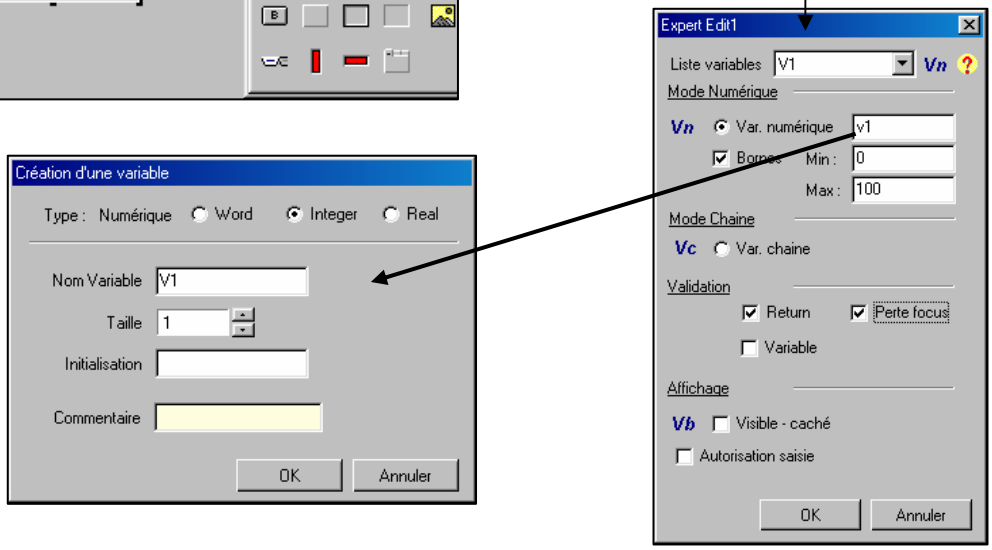
Cliquez sur le bouton, la variable **But1** passe à 1 et anime le cercle en clignotant.
Un second clic de la souris repasse le bit **But1** à False et anime le cercle d'une autre couleur en fixe

7 - Création d'un champ de saisie



Sélectionnez le symbole *champ de saisie* dans la boite à outils puis tracez le dans l'écran.

Double cliquez sur celui-ci.
Une boite de dialogue apparaît.



Cliquez sur le radio bouton **Var. Numérique**, afin de pouvoir saisir en exécution une valeur numérique.
Ici nous désirons saisir une valeur vers le variable V1 cependant cette variable n'existe pas.
Une boite de dialogue apparaît permettant ainsi la création de cette variable.

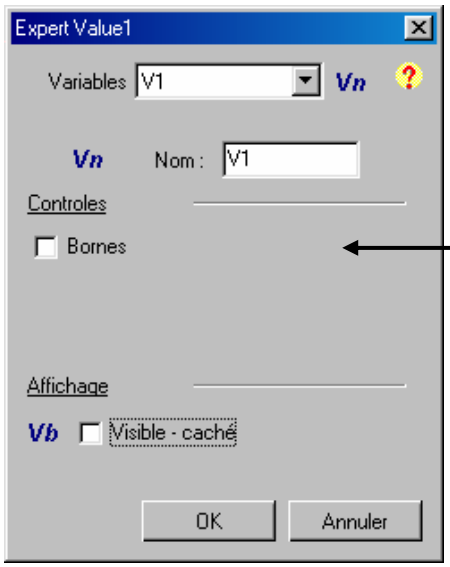
3 Types sont proposés **Word**=16 bits, **Integer**=32 Bits ou **Real**=Flottant.

Comme toutes les autres variables, une initialisation de celle-ci est possible et le nom de cette variable peut aussi décrire un tableau. Cette variable en saisie peut être bornée, cliquez alors sur la case à cocher Bornes. Saisissez les bornes de Mini et Maxi (Ici la saisie sera possible que dans l'intervalle 0 à 100 max.).

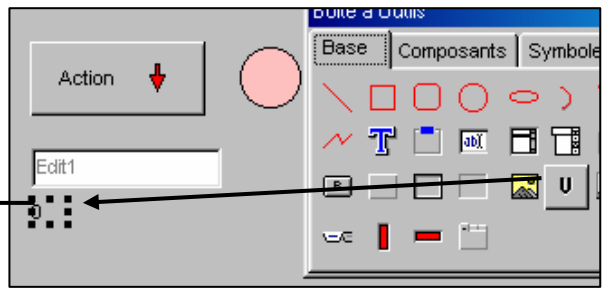
Choisissez le type d'affectation de la variable V1. Sur **Return** signifie que la valeur saisie dans le champ de saisie sera affectée à V1 sur frappe de la touche clavier <Entrée> ou sur **perte de focus** du champ de saisie (lorsque l'on passera dans champ à un autre).

8 - Affichage d'une valeur numérique

Sélectionner le composant valeur numérique dans la boite à outils, placer alors une valeur numérique sur l'écran :



Double-cliquez sur la valeur numérique placée sur le synoptique.



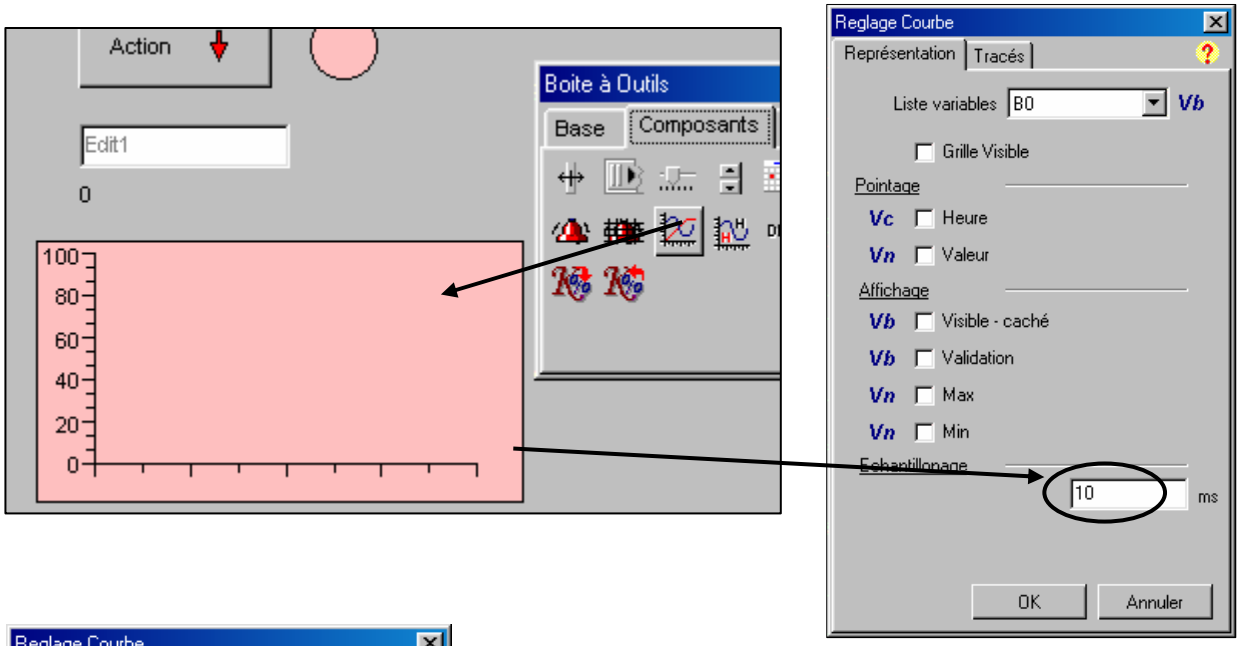
Une boite de réglage apparaît, permettant de mentionner la variable à afficher.

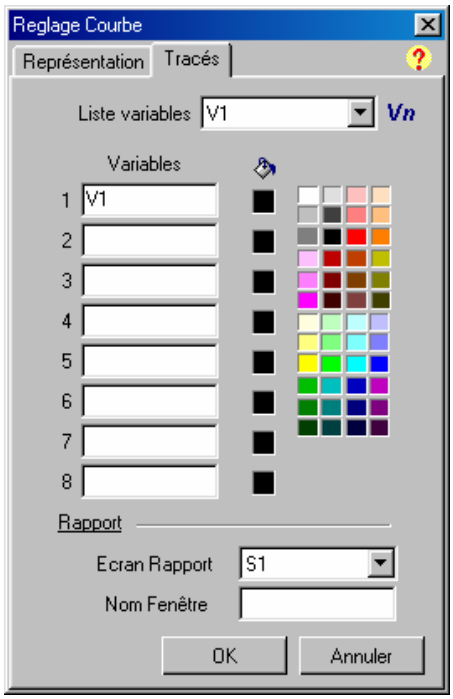
Ici choisissez la variable **V1** à afficher. Cette variable est affectée par le champ de saisie Edit1 qui à été paramétré auparavant.

Sur chaque élément graphique, une case à cocher **Visible-Caché** permet de contrôler l'affichage ou non de celui-ci en fonction de l'état d'une variable booléenne.

9 - Visualisation d'une courbe au fil de l'eau

Choisissez dans l'onglet Composants le symbole Courbes et placez celle-ci sur le synoptique.





Choisissez la période d'échantillonnage, Ici 10 ms, puis passez sous l'onglet Tracés, affectez au premier stylo, la variable V1. Chaque visionneur de courbes contient 8 Stylos (8 Courbes) maxi.

Deux cases à cocher permettent aussi d'affecter des valeurs numériques pour les Maxi et Mini en ordonnées

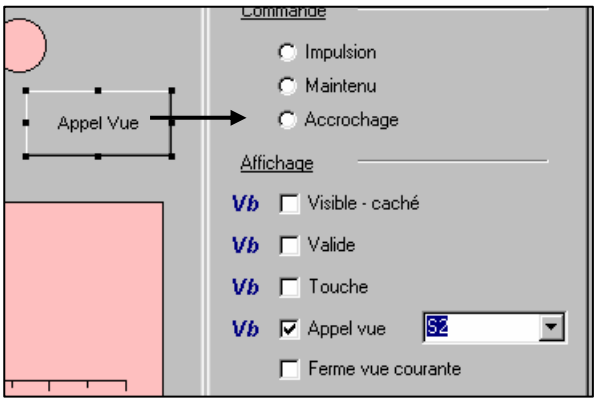
10 - Créer une application avec plusieurs synoptiques.

Visual I/O permet des applications contenant plusieurs écrans (synoptiques). Dans toute application, il existe le synoptique principal et des synoptiques secondaires. Le synoptique principal est celui qui est affiché au lancement de l'application. Pour appeler du synoptique principal d'autres synoptiques, il existe plusieurs méthodes.

1. Appel par bouton
2. Par choix dans un menu
3. Par programme.

Appel par bouton :

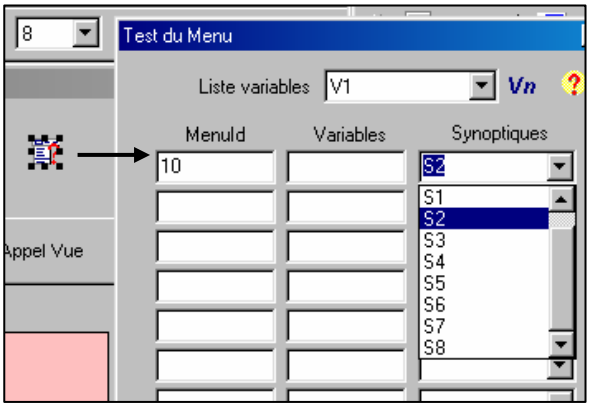
Placez un bouton sur l'écran, puis double-cliquez sur celui-ci. Une boite de dialogue apparaît :



Cochez la case appel vue, puis choisissez le nom du synoptique que le composant doit appeler.

Si le synoptique n'est pas encore créé, frappez alors son futur nom.
En plaçant un nom dans appel vue, le système rajoute automatiquement ce synoptique dans l'application finale.

Appel par choix dans un menu

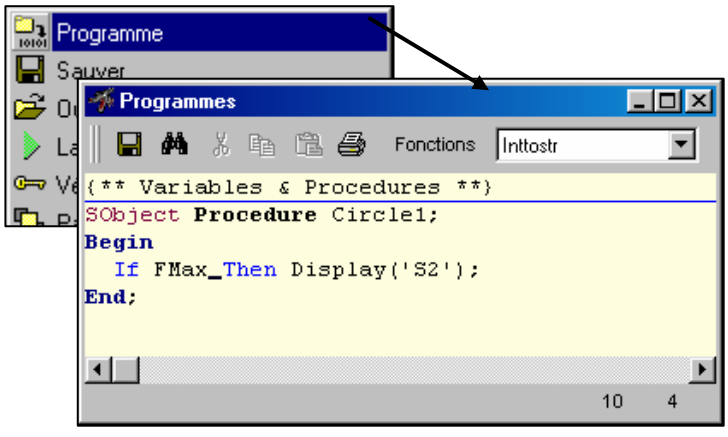


Placez le **composant Test Menu** sur le synoptique puis double-cliquez sur celui-ci.

La boite de dialogue apparaît permettant d'affecter à un identificateur de menu l'appel d'un synoptique.

Grâce à ce composant il est aussi possible d'affecter une variable Booléenne (true).

Appel par programme

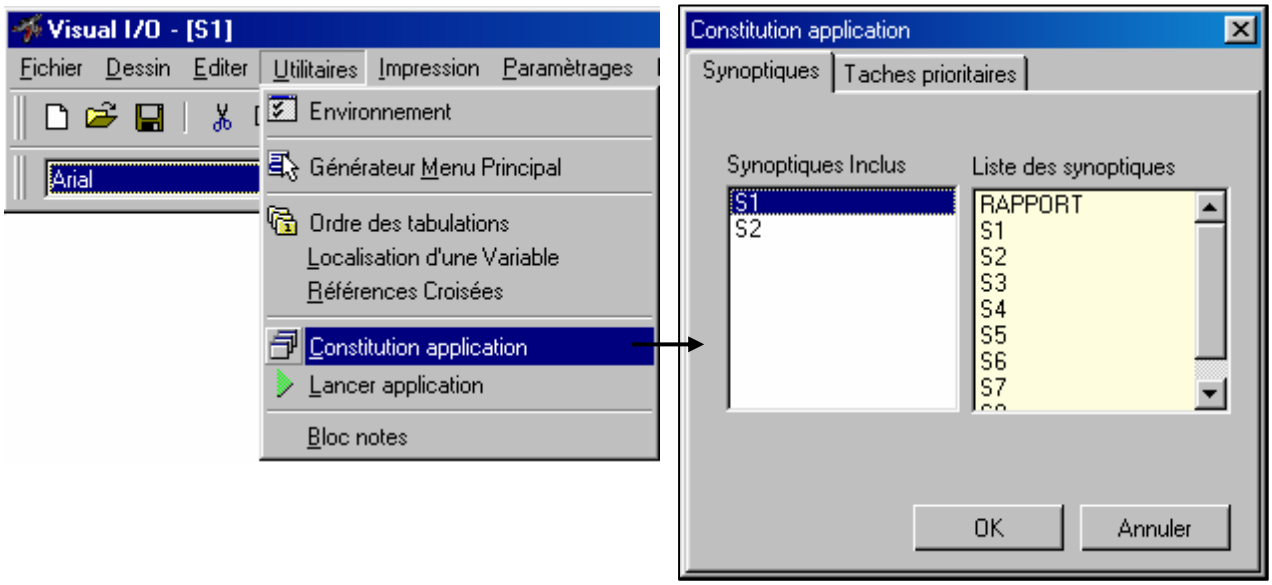


Sélectionnez une figure, puis cliquez avec le bouton droit de la souris.
Dans le menu flottant cliquez sur Programme. Une boite de code apparaît.
Frappez **If** <Variable> **Then Display**('S2').
Si la variable est vraie alors le système affiche le synoptique S2.

11 - Ajout et modifications de la liste des synoptiques de l'application

Vous pouvez ajouter, supprimer des synoptiques dans l'application finale en appelant la rubrique Utilitaires Constitution application.

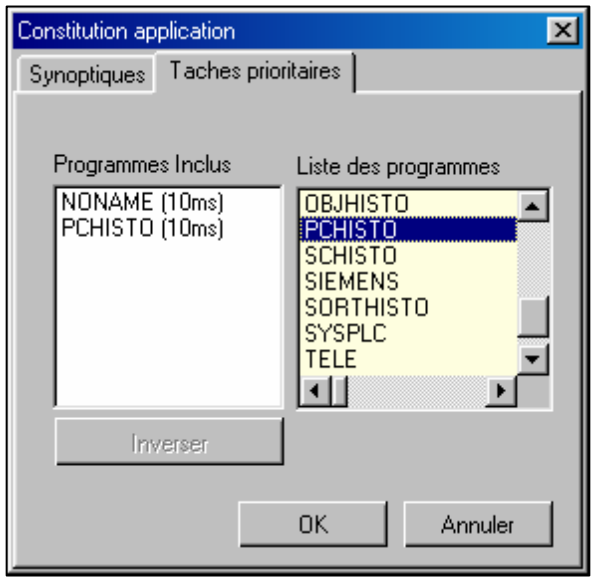
Une fenêtre apparaît :



La liste des synoptiques inclus dans l'application finale, apparaît dans la liste de gauche.
Pour **supprimer**, choisissez un nom puis frappez la touche <**Suppr**> ou **double cliquez**.

Pour ajouter un synoptique dans l'application finale, amenez par drag & drop le nom du synoptique choisi de la liste de droite vers celle de gauche ou double cliquez sur le nom à ajouter.

Sous l'onglet Taches prioritaires, le principe est le même.



2 fichiers sont générés automatiquement :

RUNVIO.PFC pour la liste des synoptiques dans l'application (Onglet Synoptiques).
PROG.OCO pour la liste des programmes fonctionnant dans le moteur temps réel de Visual I/O (Onglet Taches prioritaires).

Si vous disposez de Visual PLC, vous pouvez créer des programmes en langages ladder et grafcet puis les exécuter dans le moteur temps Réel.

Le temps d'appel des programmes est indiqué entre parenthèses.
Ce temps est modifiable dans le menu Avancés rubrique Infos VPU.

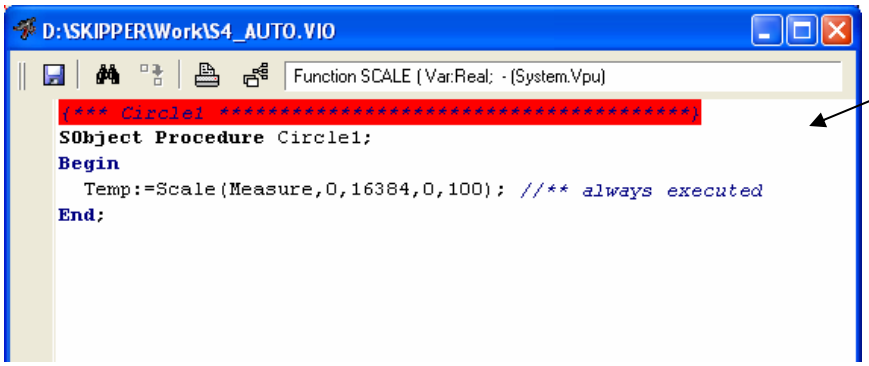
12 – Ajouter un petit programme dans mon application.

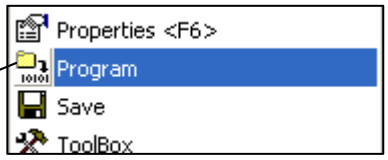
Pour ajouter un programme dans dans votre application il existe a 3 possibilités :

- Le programme est exécute seulement quand le synoptique est présent à l'écran.
- Le programme est toujours exécute quelque soit le synoptique affiché.
- Le programme est toujours exécute en tache de fond dans votre application.

Exemple: Conversion d'échelle Entrée Analogique (0 to 16384 in 0 – 100 °C)

Le programme est exécuté quand le synoptique est à l'écran
Choisissez le synoptique ; sélectionnez un objet dans celui-ci, par exemple un cercle, un boutton ou une zone edit (editbox), peu importe le composant, clickez droit avec la souris sur l'item *programme*.

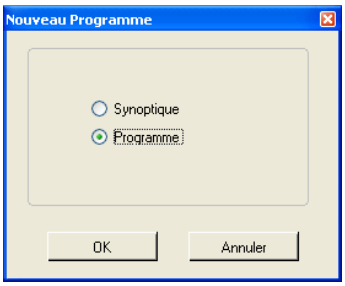


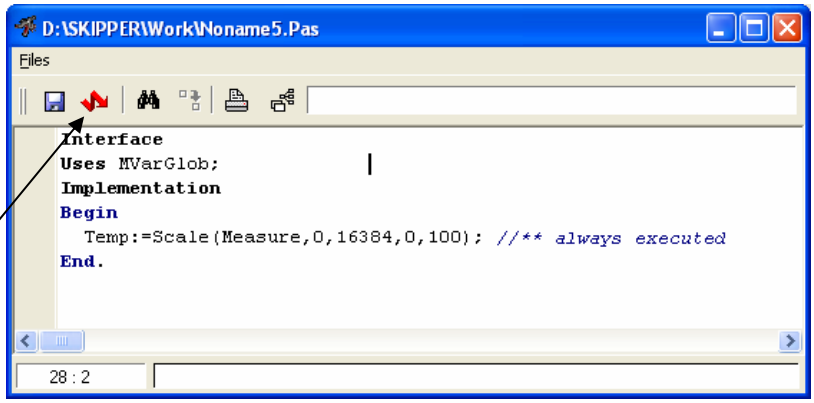


Ici la fonction **Scale** est exécutée périodiquement chaque fois que le syno S4 est visible à l'écran. Le programme n'est pas forcément en relation avec le Cercle1. Le cercle est seulement utilisé comme contenant.

Le programme est toujours execute quelque soit le synoptique affiché.
Pour cela vous devez choisir le synoptique principal. Le synoptique principal est celui qui est toujours affiché en premier au lancement de votre application. Ce synoptique est listé en première position dans la liste des synoptiques dans la rubrique constitution de l'application. (Voir menu principal).
Concernant l'éxécution, tout est absolument similaire au cas précédent .

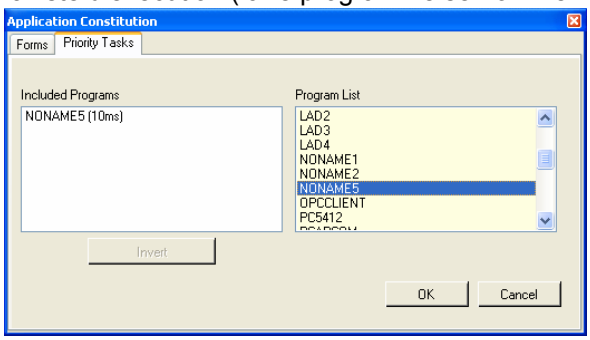
Le programme est toujours exécute en tache de fond dans votre application.
Vous devez créer un nouveau fichier contenant ce programme (dans notre exemple ci-dessous, c'est simplement une seule ligne de code). Cliquez sur le bouton Nouveau puis choisissez Programme. Un nom de programme est automatiquement généré et un éditeur de texte apparaît.





*Après avoir écrit votre programme, vous devez le compiler en clickant sur le bouton **Zigzag**.*

Normalement l'éditeur affiche un modèle vierge avec écrit **Unit** sur la première ligne. Vous devez effacer cette ligne pour transformer le code en programme cyclique. L'expression la plus simple de ce programme est d'écrire le code programme entre **Begin** et **End** appellé *corps du programme*. Ce corps de programme sera périodiquement appellé par le système (toutes les 10ms).
Après avoir compilé, vous devez ajouter ce programme dans la liste d'exécution (ici le programme se nomme Noname5).



Pour ajouter dans la liste d'exécution (liste de droite)
les programmes compilés, ammenez par glissé/déplacé
ou double click le programme choisi dans la liste de droite.

Pour les details, voir les chapitres

- Programmation
- Creation de librairiesVPU ou programmes cycliques

Types de variables

Dans Visual I/O, vous pouvez utiliser différents types (système) de variables :

Types numériques

Chiffres sans virgules

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

Chiffres avec virgules

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

Type chaînes de caractères

String : Chaîne de 255 caractères. (De 1 à 255 caractères) 256 octêts. *Ex : S :='ARSOFT' ;*

Type état

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

Visual I/O permet l'adressage Bit dans un mot. *Ex : W10.0 :=False (Premier bit du mot W10).*

Type tableau

Array : Tableau de type de variables définies ci-dessus. *Ex : Array [0..12] of Boolean;*

Définir ses propres types

Vous pouvez redéfinir le nom d'un type de base ou créer vos propres types.

```
Type   // *** Type est un mot réservé du langage *****
  Handle : Integer ;
  Bool   : Boolean ;

// *** Section var d'un programme ou d'une librairie permet la déclaration
Var
  H  : Handle;  // *** équivaux à H : Integer;
  WB : Bool;
```

Type Record

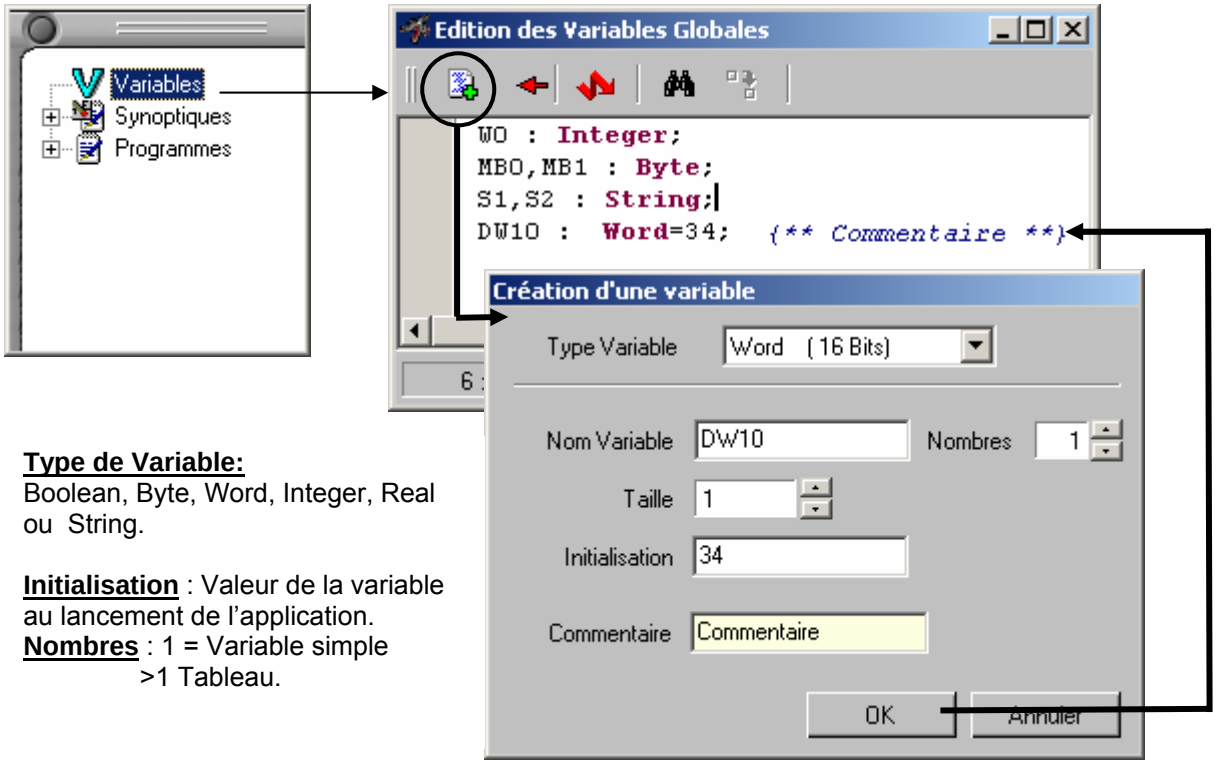
```
Type
TPoint = Record
          X : Integer;
          Y : Integer;
         End;
Var
 Point : TPoint;

// *** Exemple d'utilisation *****
Begin
  Point.X:=12;
  Point.Y:=56;
End;
```

Créer des variables globales

Appeler l'éditeur de variables globales

Dans le navigateur, double-cliquez sur *Variables*. Un éditeur de texte apparaît :

Vous pouvez alors frapper directement les noms de variables ainsi que leurs types ou bien utiliser l'expert permettant d'ajouter à la suite des variables.



Type de Variable:
Boolean, Byte, Word, Integer, Real ou String.

Initialisation : Valeur de la variable au lancement de l'application.
Nombres : 1 = Variable simple
>1 Tableau.

Compilation des variables globales

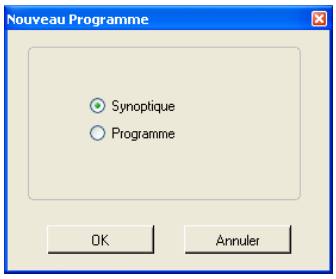
Fichiers créés

Fichier source : Déclaration des variables **MVARGLOB.PAS**
Fichier final : Après compilation du fichier source Mvarglob.pas création du fichier **MVARGLOB.VPU**

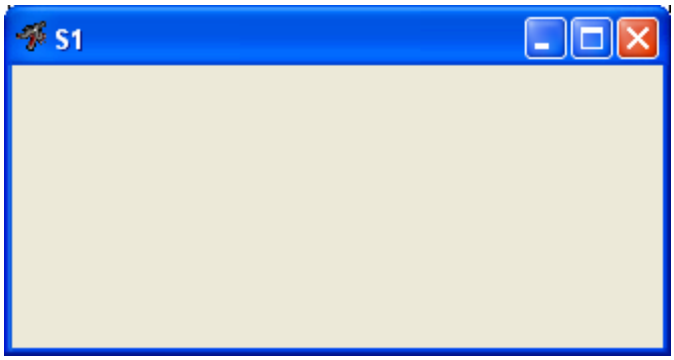
Après chaque compilation la barre de status vous informe si celle-ci c'est bien effectuée.

Créer un nouveau synoptique

Dans le menu *Fichiers*, cliquez sur *Nouveau* puis choisissez *Synoptique* puis cliquez sur le bouton *OK*

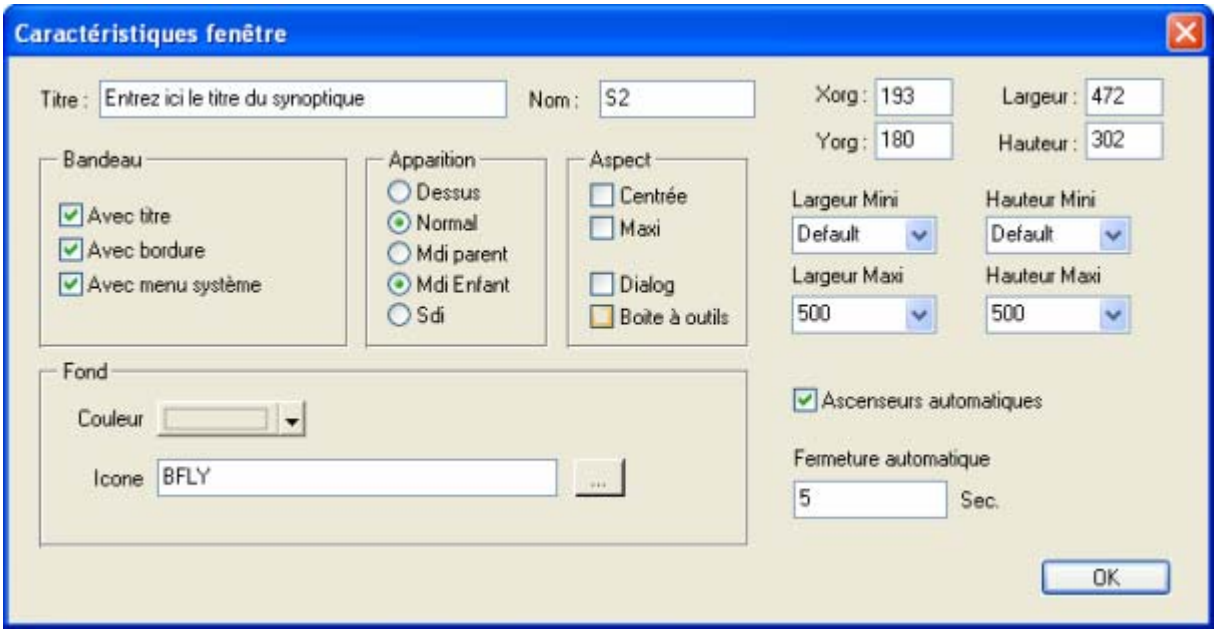


Un synoptique vide apparaît de nom S1, puis S2,S3 etc.. ►



Caractéristiques d'un synoptique

Double cliquez sur le fond du synoptique ou dans le menu flottant *Description Synoptique*



Dans le champ *Titre*, entrez le nom titre apparaissant dans la barre de titre du synoptique.
Dialog et boite à outils ont des fonctionnalités concernant les programmes du synoptique.
En cochant une de ces options, les programmes inclus dans ce synoptique seront exécutés que si la fenêtre à le focus (barre de titre active).
Sdi permet dans d'avoir des synoptiques flottant sur l'écran, tandis que **Mdi** doit avoir uniquement une fenêtre parent et des synoptiques enfants.
Les synoptiques enfants seront enfermés dans la fenêtre parent et ne pourrons pas en sortir. Le mode SDI est le plus utilisé en général. L'option **dessus** permet que le synoptique ne soit pas recouvert par d'autre (qu'il reste en avant plan).

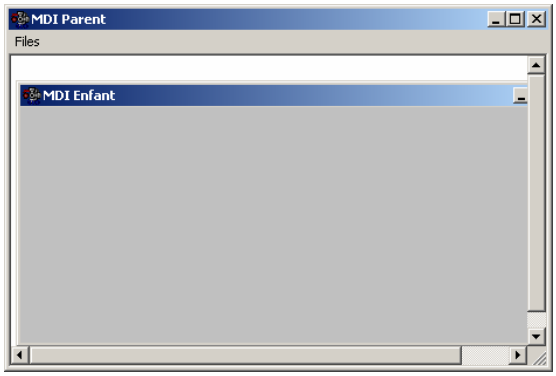
Titre	Défini le titre apparaissant dans le bandeau de la fenêtre.
Nom	Défini le nom de fichier graphique et programme (.syn, .Vio). Modifiable par enregistrer sous ou par renommer dans le navigateur de Visual I/O.
Avec Titre	Permet l'affichage de la fenêtre avec ou sans titre.
Avec Bordure	Permet l'affichage de la fenêtre avec ou sans bordure permettant de retailler celle-ci.
Menu Système	Défini ou non un menu système dans cette fenêtre. Le menu système est les petits boutons en haut à droite du bandeau de la fenêtre.
Xorg,Yorg	Coordonnées absolues du coin haut gauche de la fenêtre sur l'écran.
Largeur Hauteur	Largeur et Hauteur du Synoptique courant.
Fond	Apple la palette de Couleurs pour fixer la couleur de fond du synoptique.
Icône	Permet de choisir une icône pour ce synoptique.

Largeur Mini Permet de définir la largeur **Minimum** autorisée lors d'un retaillage de la fenêtre.
Les valeurs autorisées sont valeur numérique, None ou défault.
Hauteur Mini Permet de définir la hauteur **Minimum** autorisée lors d'un retaillage de la fenêtre.
Les valeurs autorisées sont valeur numérique, None ou défault.
Largeur Maxi Permet de définir la largeur **Maximum** autorisée lors d'un retaillage de la fenêtre.
Les valeurs autorisées sont valeur numérique, None ou défault.
Hauteur Maxi Permet de définir la hauteur **Maximum** autorisée lors d'un retaillage de la fenêtre.
Les valeurs autorisées sont valeur numérique, None ou défault.
Valeur numérique en pixel (>0)
None : Aucune restriction de taille
Default : La valeur actuelle (valeur du champ largeur).
Ascenseurs automatiques Permet d'afficher les ascenseurs horizontal et vertical en cas de retaillage de la fenêtre. Si cette option n'est pas cochée aucun ascenseur ne sera affiché.
Fermeture automatique Permet en run de fermer automatiquement la fenêtre si aucun événement clavier ou souris ne survient ceci au bout d'une temporisation déterminée en secondes.

Type de fenêtres

Apparition :

Dessus Permet au synoptique courant d'apparaître toujours en premier plan.
Lorsque plusieurs fenêtres sont à l'écran et quelles se chevauchent, le fait de cliquer sur une fenêtre la fait passer en premier plan. Si le synoptique à l'attribut Dessus coché, il ne pourra pas être recouvert par une autre fenêtre de l'application. Ceci est le cas de la boite à outils de l'éditeur Visual I/O.
Normal Permet au synoptique de faire recouvrir par une autre fenêtre c'est le cas inverse de l'option Dessus.
MdiParent Les applications Visual I/O peuvent être des applications MDI ou SDI.
Une application MDI à besoin d'une fenêtre Parent et une seule puis de multiples fenêtres Enfants. Cocher cette option indique que votre synoptique est le principal et qu'il est la fenêtre Parent d'une application MDI. Si plusieurs parents sont paramètrés dans une application le premier sera véritablement le parent. Les autres seront attribués enfants par le système.
MdiEnfant Défini votre synoptique comme un enfant d'une application MDI.
Si celui-ci contient un menu principal, celui-ci remplacera le menu parent lors de son affichage.
Sdi Votre synoptique ne fait pas partie d'une application MDI. il n'a donc pas de fenêtre Parent.
Un synoptique SDI peut être inclus dans une application MDI.
Ce synoptique ne respectera pas les règles MDI il pourra alors sortir du cadre de la fenêtre parent.
Centré Cette option permet au synoptique de se centrer tout seul au milieu de l'écran au premier affichage.
Maxi Défini le synoptique dans la taille maximum possible (tout l'écran).
Dialog Défini le synoptique comme boite de dialogue.
Cela permet d'afficher une présentation différente de la fenêtre. Cela permet d'exécuter les programmes internes à ce synoptique que si la fenêtre a le Focus (barre de titre active).
Par soucis d'optimisation de la vitesse d'exécution de l'application, passer des synoptiques en Dialog permet de répartir le temps machine. Certains programmes de synoptique n'ont pas besoin d'être exécutés si la fenêtre est présente mais pas en active.
Boite à Outils Cette option permet d'afficher une présentation du Synoptique comme une petite boite à outils. Le bandeau principal est diminué et seulement une commande de fermeture apparaît dans celui-ci. Cette option reprend les caractéristiques en programme de l'option Dialog.



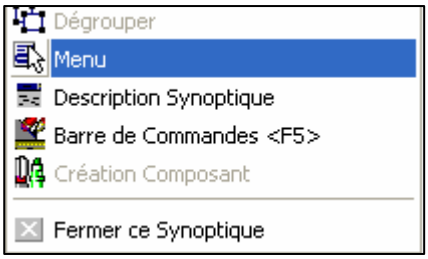
***Application MDI**, la fenêtre principale est un conteneur de fenêtres enfants.*

***Application SDI**, la fenêtre principale et les autres fenêtres de l'application sont libres de se déplacer sur la totalité de l'écran.*

Générer un menu principal

Il est possible d'ajouter un menu à vos synoptiques.
Si vous créez des applications SDI (Documents à interface simple) avec différents synoptiques ayant leurs propres menus, ceux ci seront toujours rattachés à leurs synoptiques respectifs.
Dans des applications MDI (Documents à interfaces multiples), le synoptique Maître contient un menu et les menus des synoptiques enfants (s'ils existent) viennent remplacer le menu principal lors de leurs affichages.

Pour la création d'un menu principal d'un synoptique appelez la commande 'Menu' du menu flottant (clic droit de la souris).



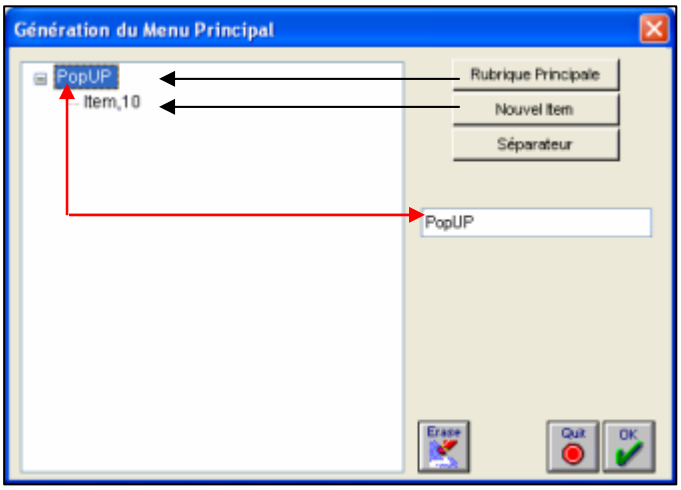
Une boite apparaît avec à gauche une ListBox vide .
Cette ListBox symbolisera sous forme d'arborescence les différents éléments du menu principal.

Chaque rubrique principale pourra contenir des Eléments appelés Items.
Dans ces rubriques principales nous allons pouvoir insérer des sous rubriques contenant-elles aussi des items et ainsi de suite.

Les items sont des textes qui apparaîtrons sous la rubrique parent. Outre les items nous allons pouvoir insérer aussi des séparateurs (Traits horizontaux sans texte).
Les items aurons des numéros d'identification permettant au programme de tester si l'item cliqué est celui que l'on attend. Certaines commandes tel que **SetMenuString** ou **ToggleMenu** permettent de modifier en Run (par programme) l'aspect de ces items.
Lors de la création d'un nouvel item un numéro unique lui est attribué. Cependant ce numéro est modifiable mais il doit être unique afin de pouvoir tester en Run l'item à l'aide du composant test Menu par exemple.
Les titres des rubriques n'ont pas d'identificateurs. En Run pour les adresser (commande SetPopupMenu) nous admettons la convention suivante la rubrique la plus à gauche porte le numéro 0. Sa voisine de droite porte le numéro 1 Etc.

Les rubriques ou les items peuvent avoir des lettres d'appel (raccourcis) comme par exemple la rubrique Fichier a la lettre F pour appel.
Pour déterminer la lettre d'appel placez le caractère & devant.
Le fait de placer le caractère & devant la lettre choisie à pour effet de souligner la lettre et de pouvoir appeler en 'Run' directement la rubrique en conjuguant la touche <ALT> + <la lettre choisie>.
Par exemple pour appeler la rubrique Fichier sans cliquer avec la souris frappez simultanément <Alt><F>.

Fenêtre de création d'un menu.



Rubrique Principale
Ajoute une nouvelle rubrique à la suite de celles existantes.

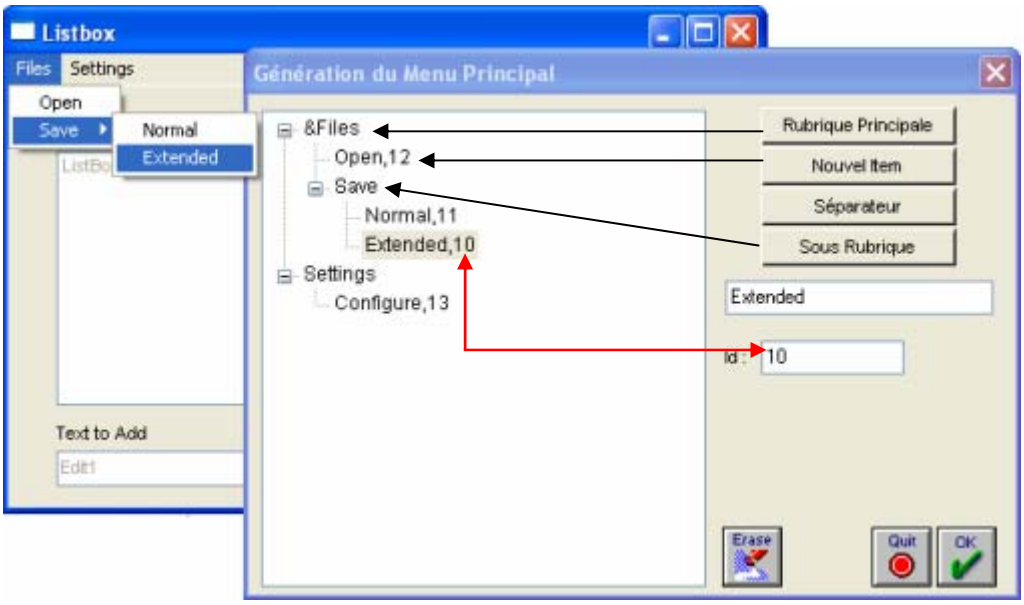
Nouvel Item
Ajoute un item après la position du bandeau.

Séparateur
Ajoute un séparateur après la position du bandeau.

Sous rubrique
Transforme un item en une sous rubrique.

A chaque ajout ou suppression, le menu principal de votre synoptique est remis à jour.

Exemple de menu principal



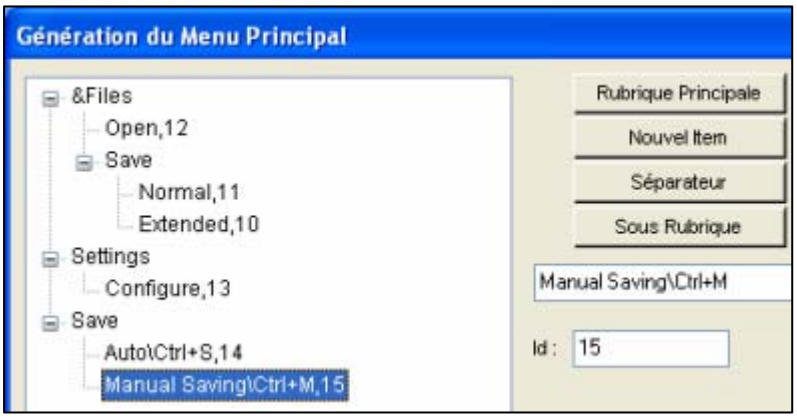
&Files est une rubrique. Le caractère & devant la lettre F permet de souligner celle-ci et de déterminer les touches ALT +F comme touches de raccourci.
Les rubriques n'ont pas d'identificateurs seul le texte est modifiable dans cette boite.

Open,12 est un item de texte Open et de valeur d'identificateur égal à 12. Cet identificateur sera testé par le programme utilisateur ou un composant pour déterminer l'action sur le menu qui aura été effectué.

Save est un item transformé en sous rubrique par le biais du bouton Sous rubrique.

Les identificateurs sont modifiables par le champ de saisie Id :

Alignement des textes des items

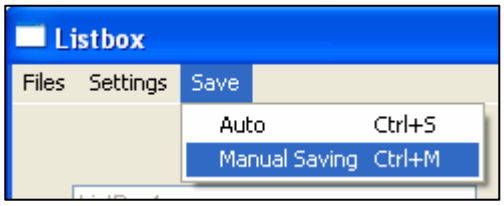


Il est possible de faire apparaître un texte complètement à droite d'un Item de menu en insérant le caractère \ devant ce texte.

Exemple

Auto\Ctrl+S

Manual Saving\Ctrl+M

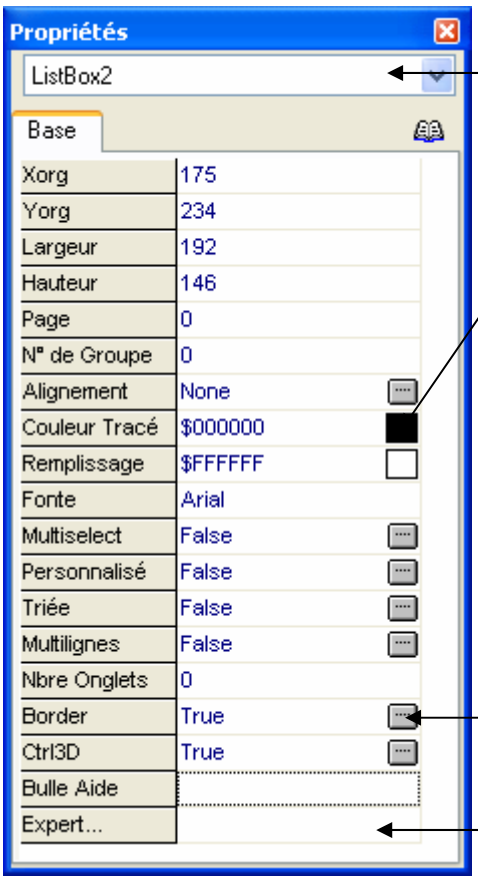


☞ Attention : Ne pas utiliser deux fois les mêmes identificateurs d'item dans une application comportant plusieurs synoptiques avec des menus déroulants.

Boite de propriétés graphiques

La boîte de propriétés graphiques d'une figure est appelée en sélectionnant la figure (cliquez dessus !) puis en frappant la **touche <F6>**. Une fenêtre apparaît :

Champs de saisies



Le ComboBox contient la liste de toutes les figures présentes sur le synoptique. Choisissez une figure pour visualiser ses attributs.

Une palette de couleur personnalisable permet de fixer la couleur de contour et de remplissage d'une figure. Les couleurs système peuvent être utilisés.

Dans le cas de certaines figures, des propriétés supplémentaires peuvent aussi apparaître à la suite des propriétés de base.

Pour les objets type fenêtre (listbox, editbox, window) le contour (border) et type de contour (3D,Normal) peuvent être modifiés.

Vous pouvez appeler l'expert en cliquant sur cette cellule

Xorg,Yorg	Coordonnées coin haut gauche de la figure.
Largeur,Hauteur	Largeur et hauteur de la figure sélectionnée.
Couleur Tracé	Couleur de contour de la figure. Double cliquez pour choisir la couleur dans la palette située sur la droite.
Style Tracé	Style de tracé des figures dérivant du point (lignes,Rectangles).
Epaisseur	Style du tracé des figures dérivant du point (lignes,Rectangles).
Remplissage	Couleur de remplissage de la figure. Double cliquez pour choisir la couleur dans la palette située sur la droite.
Remplie	Figure remplie ou non. Double cliquez pour basculer cette case à cocher.
Fonte	Fonte d'écriture d'une figure contenant du texte. Double-cliquez pour l'appel de la fenêtre de fontes disponibles.
Angle	Permet de modifier l'angle d'écriture d'un texte (Fonte TT seulement).
Page	Permet de modifier le numéro de page contenant la figure sélectionnée (pour compatibilité) .
Nbre Onglets	Permet de régler le nombre d'onglets d'un Tabset ou de colonnes dans une liste box.
N° Groupe	Dans le cas ou la figure est groupée avec d'autres le numéro de groupe alloué par Visual I/O apparaît. Si la figure n'est pas groupée le numéro 0 est inscrit.
Message	Texte d'un Bouton ou d'un texte. Nom du fichier Bitmap,Méta fichier ou Jpeg à afficher. Double Cliquez pour faire apparaître une boite de dialogue de sélection et de prévisualisation.
Decimales	Nombre de décimales pour l'affichage d'une valeur numérique (V). Nombre d'emplacements pour l'affichage des onglets dans une boîte à onglets.
Multilignes	Utilisé exclusivement pour les champs de saisie (EditBox). Indique si le champ de saisie est multi lignes ou simple ligne.

H_Scroll Barre de Scroll horizontal pour un champ de saisie multi lignes.
Centrage automatiquement à droite d'un texte. Si cette option est vraie (True) lors de la modification par programme d'un texte, celui-ci s'alignera toujours sur la droite.
Sinon l'alignement est effectué sur gauche.

V_Scroll Barre de Scroll vertical pour un champ de saisie multi lignes.

Mot de Passe Si cette option est vraie (true) l'affichage et la saisie des caractères dans la zone de saisie sera
remplacé par des caractères *.

Multiselect Autorise ou non la sélection multiple dans une listeBox.

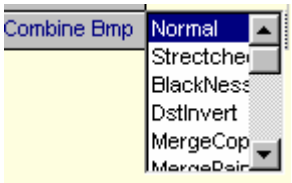
Type Combo Fixe le type de ComboBox (1 : Simple, 2 : DropDown, 3 : DropDownList).

Personnalisé Fixe la personnalisation ou non d'une ListBox ou d'un ComboBox.
La personnalisation permet d'afficher autre chose que du texte dans par exemple un ComboBox ou une ListBox. L'instruction DriveComboBox affiche la liste des unités de disque de votre PC à l'aide de bitmaps et de textes. Les procédures CustomDrawItem, CustomMeasureItem seront utilisés pour l'affichage personnalisé ou tout simplement la procédure AddString.

Trié Met en place le trie automatique dans une ListBox ou un ComboBox. Les ComboBox et ListBox personnalisées ne peuvent être triées.

Propriétés d'une image Bitmap

Combine Bmp Permet une combinaison graphique du bitmap à l'aide des attributs listés.
Les attributs les plus utilisés sont normal et Stretched.



Normal Affiche le Bitmap dans sa taille originale**.**

Stretched Agrandi le Bitmap au cadre tracé.

Alignements

Détermine comment le contrôle s'aligne dans son conteneur (ou contrôle parent).

Le premier permet d'agrandir une figure pour s'adapter à la nouvelle taille du synoptique. Une fenêtre contenant une grille peut par exemple avoir besoin de cette fonctionnalité. (D-Right, D-Bottom, D-Right/Bottom, None).

Le deuxième permet de déplacer automatiquement une figure lors d'un retaillage du synoptique par les bordures de celui-ci. Cela permet par exemple de toujours visualiser un bouton quelle que soit la taille du synoptique. (Right, Bottom, Right/Bottom, None).

Le troisième permet de positionner un élément graphique en taille et position en haut en bas à droite à gauche ou dans la zone libre dite 'Client'. (Altop, AlBottom, AlRight, AlLeft, AlClient).

☞ **Note** : Permet d'annuler tout alignement. Ce qui signifie qu'un contrôle reste là où il a été placé dans la fiche ou le volet.

Aligner un contrôle en haut, en bas, à gauche ou à droite d'une fiche ou d'un volet de manière à ce qu'il reste à cet emplacement même si la taille de la fiche, du volet ou du composant contenant ce contrôle change. Lorsque le parent est redimensionné, un contrôle aligné modifie aussi ses dimensions afin de continuer à s'étendre vers le côté supérieur, inférieur, gauche ou droit du parent.

Par exemple, pour utiliser comme palette d'outils un composant volet contenant divers contrôles, attribuez la valeur AlLeft à la propriété Align de ce volet. Cette valeur garantit le positionnement de la palette d'outils sur le bord gauche de la fiche avec une hauteur égale à celle de la hauteur client de la fiche.

Un nombre quelconque de composants enfant d'un même parent peuvent avoir la même valeur pour Align. Dans ce cas, ils s'empilent le long du bord du parent. Les contrôles enfant sont empilés dans l'ordre z

Déplacements

None	la figure ne change pas de place en cas de modification de la taille du synoptique. Eventuellement une barre de scroll apparaît sur les bords du synoptique.
D-Right	En cas de retaillage du synoptique, la figure se déplace pour toujours rester placée à la même distance du bord droit du synoptique.
D-Bottom	En cas de retaillage du synoptique, la figure se déplace automatiquement pour toujours rester placée à la même distance du bord bas du synoptique.
D-Right/Bottom	Combinaison des attributs D-Right+D-Bottom.

Agrandissements

Right	La figure s'agrandie pour que son bord droit reste toujours à égale distance du bord droit du synoptique.
Bottom	La figure s'agrandie pour que son bord bas reste toujours à égale distance du bord bas du synoptique.
Right/Bottom	Combinaison des attributs Right+Bottom.

Taille et Position

AlTop	La figure se positionne en haut du synoptique et prend toute la largeur de celui-ci.
AlBottom	La figure se positionne en bas du synoptique et prends toute la largeur de celui-ci.
AlLeft	La figure se positionne à gauche du synoptique et prends toute sa hauteur.
AlRight	La figure se positionne à droite du synoptique et prends toute sa hauteur.
AlClient	La figure se positionne en général au centre ou dans la zone restante en fonction des autres figures ayant les attributs AlTop alBottom AlRight et AlLeft.

☞ ***Note :*** Les figures ayant un de ces attributs modifient leurs tailles en Run en fonction d'un agrandissement ou d'une réduction du synoptique par l'utilisateur.

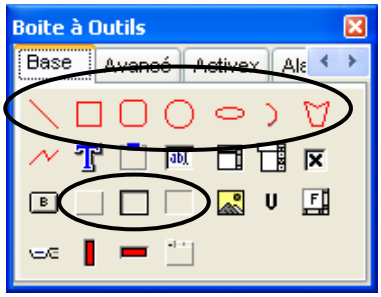
Agrandissement des objets

Sélectionnez les figures
Frappez simultanément <CTRL> <+> pour agrandir les figures sélectionnées.
Frappez simultanément <CTRL> <-> pour réduire les figures sélectionnées.

Les Experts des figures graphiques.

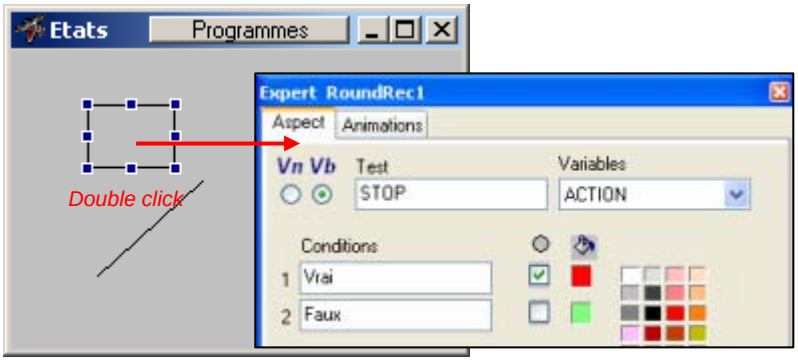
Les boites de dialogues expert permettent d'affecter des traitements prédéfinis par simples options et renseignements de champs de saisie.
Ligne, Rectangle, Cercle, Ellipse, Arc, Polygone, Supports

Ligne, Rectangle, Cercle, Arc, Ellipse, Polygone, Supports



Description de l'expert

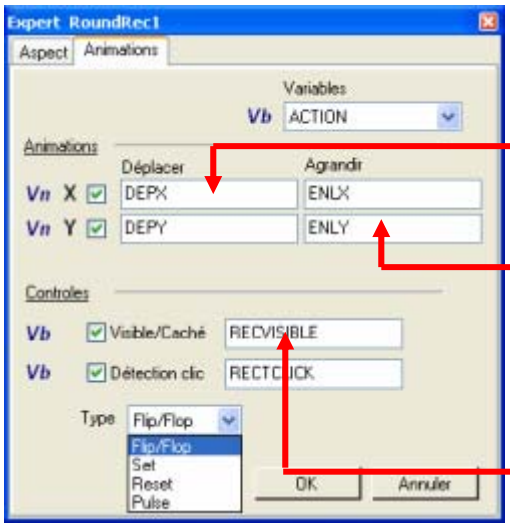
Coloriage intérieur du rectangle en fonction d'un bit.



Clignotant Rouge/Transparent si Stop est vrai.

Coloriage intérieur en vert lorsque Stop est faux.

Déplacement et agrandissement d'un rectangle en fonction de variables numériques



Déplacement du rectangle en relatif par rapport à la position originale.

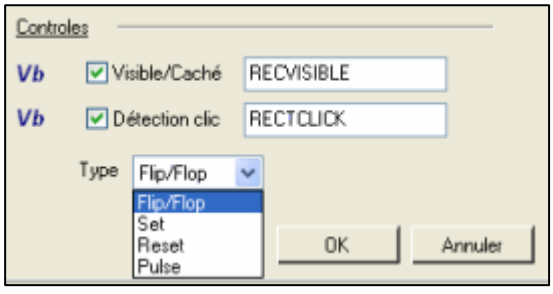
Déplacement en X en fonction de la variable **DepX** (en pixels).
Déplacement en Y en fonction de la variable **DepY** (en pixels).

Agrandissement du rectangle en relatif par rapport à la taille originale.

Agrandissement en X en fonction de la variable **ENLX** (en pixels).
Agrandissement en Y en fonction de la variable **ENLY** (en pixels).

Le Rectangle est visible ou caché en fonction de la variable booléenne RecVisible.

Créer un bouton à l'aide d'un rectangle



En cliquant dans le rectangle, il est possible de le transformer en bouton de type Pulse, reset, Set ou Filp/Flop.
Le bit RectClick est affecté en fonction du type de traitement choisi.
Si Flip/Flop est le type choisi, un click gauche avec la souris sur le rectangle, passera le bit RetClick à True, un second clic passe le bit RetClick à false.

Coloriage intérieur du rectangle en fonction d'une valeur numérique

◄ L'autre type de test est une comparaison numérique. Cochez le radio bouton ***Vn***, 8 tests sur une variable numérique sont possibles.

La variable testée n'est plus une variable booléenne mais une variable numérique (Word, Integer ou Real).

8 Conditions sont possibles.
A chaque condition une couleur est affectée à la figure avec ou sans clignotement.
La condition à tester peut aussi faire référence à la valeur d'une autre variable numérique.

Ici la variable DEPY est comparée aux valeurs 10, 20, 30 ou W0 afin de colorier le rectangle en conséquence

☞ **Note** : Seul pour le **Polygone** ou **Polyline** la **rotation** est acceptée. Cette rotation s'effectue selon un angle donné contenu dans la variable spécifiée.

La rotation s'effectue dans le sens des aiguilles d'une montre. Rendant ainsi plus aisée la représentation d'aiguilles dans des cadrans.

Seul pour ces 2 types de tracés, un champ de saisie supplémentaire apparaît.

Tester un bit dans un mot

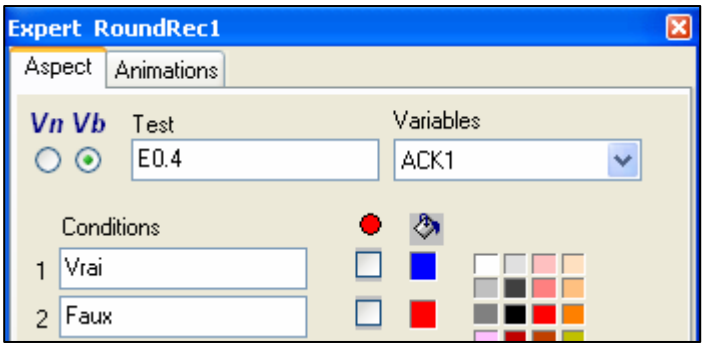
Avec Visual I/O il est très facile de tester ou d'affecter un bit dans un mot qu'il soit Byte, Word ou integer. Il suffit d'indiquer le mot suivi d'un point puis le bit souhaité.

Le premier bit est 0 le dernier d'un integer est le Bit 31.

Exemple Test du Bit 0 du Mot W10 `If W10.0 Then` (Etat True ou False comme un booléen).

Exemple : Affectation d'un bit dans un mot :

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



Test du bit 5 du mot E0 si à 1 coloriage du rectangle en Bleu sinon en Rouge.

Programmation sur les rectangles, cercles, lignes et polygones.

```
// *** RoundRec1 ************************************
SObject Procedure RoundRec1;
Begin
  // *** Programmes concernant le Rectangle RoundRec1
End;
```

Changement de couleur de remplissage en fonction d'un bit.

```
If Stop Then SetFillColor(255) ;  // ** Colorie en Rouge si stop est vrai
If Stop=False Then SetfillColor($00FF00); // ** Colorie en vert si stop en faux.
```

Changement de couleur de contour en fonction d'un bit.

```
If Stop Then SetColor(255) ;    // ** Colorie en Rouge si stop est vrai
If Stop=False Then SetColor(0); // ** Colorie en noir si stop en faux.
```

Test de click souris dans le rectangle affichage du nom de la figure.

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

Afficher et cacher en fonction d'un bit

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

Test si la souris est sur la figure

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

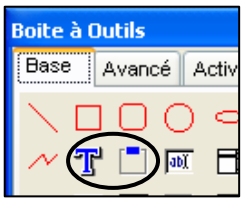
Test l'arrivée et le départ de la souris d'une figure

```
If Arrive then
Begin
  Commut_Fig('Text1'); //** Change de figure adresse le text1
  SetText('Arrive');   //** Change le texte de la figure Text1
End Else
If Depart Then
Begin
  Commut_Fig('Text1');
  SetText('Depart');
End;
```

Mise en mémoire des coordonnées d'une figure

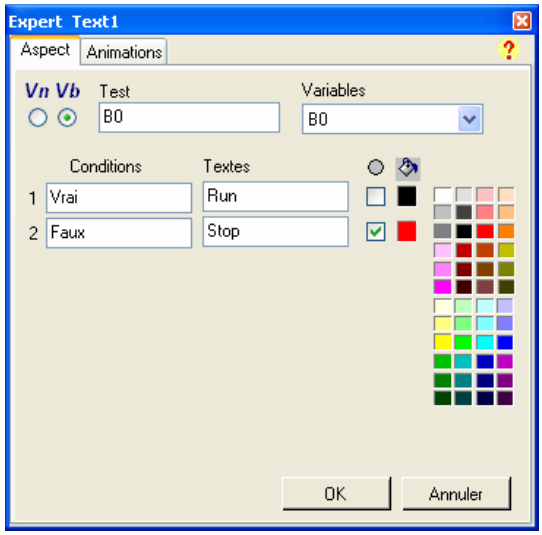
```
// *** RoundRec1 ************************************
SObject Procedure RoundRec1;
Var
  Xorg, Yorg    : Integer;
  Width, Height : Integer;
Begin
  // *** At the first display *****
  If Start_Visu Then
  Begin
    Xorg  :=GetXorg;
    Yorg  :=GetYorg;
    Width :=GetXFin-Xorg;
    Height:=GetYFin-Yorg;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

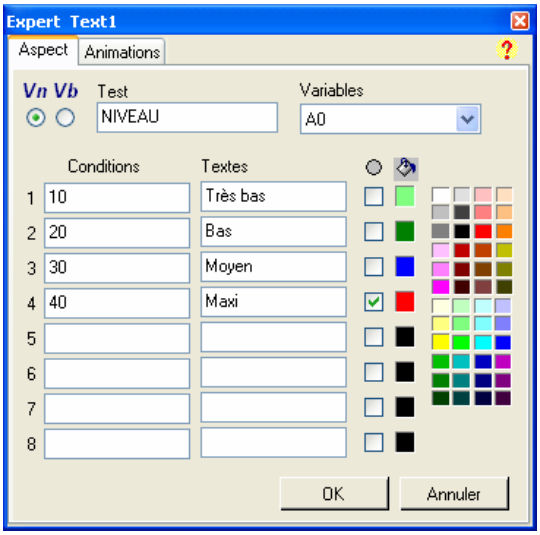
Texte, Groupe



Cet expert permet le changement ou le coloriage d'un texte ainsi que son déplacement graphique sur l'écran.

2 types de traitements sont possibles :
Le test d'une variable booléenne avec ses 2 conditions **possible Vrai ou Faux**.
Selon l'état, le texte sera colorié dans la couleur paramétrée et un nouveau texte sera affiché. Si la case à cocher (dessous cercle clignotant) est validée, le texte clignotera quand la condition sera vérifiée. Amenez la variable désirée du ComboBox *Variables* vers le champ de saisie *Test*. ou frappez le nom de la variable à tester.
Si cette variable n'existe pas, le système la créée automatiquement pour vous en la rajoutant dans la liste des variables globales.

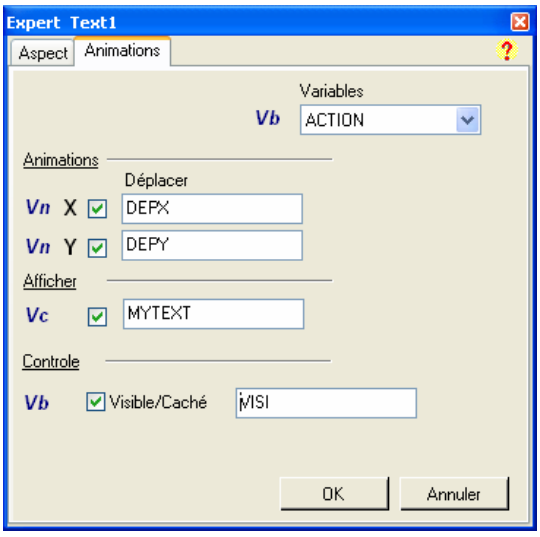




L'autre type de test est une comparaison numérique.
La variable testée n'est plus une variable booléenne mais une variable numérique (Word, Integer ou Real).
8 Conditions sont possibles.
A chaque condition une couleur est affectée au texte avec ou sans clignotement ainsi qu'un nouvel intitulé.

La condition à tester peut aussi faire référence à une autre variable numérique.

Animations Texte, Groupe



En fonction du champ de saisie, la liste des variables du ComboBox '*Variables'* change.

Pour la rubrique Animations seul les variables numériques sont autorisées.

Dans le champ *Déplacer* en X ou Y indiquez la variable contenant le déplacement relatif aux coordonnées d'origine.

Ex : Si la variable DepX contient 10, alors le texte se déplacera de 10 Pixels en X par rapport à sa position d'origine que vous aviez tracé sous l'éditeur.

L'option **Afficher** permet d'afficher à l'écran le contenu d'une variable Chaîne de caractères. Ici le texte sur l'écran affichera constamment le contenu de la variable Chaîne *Selec*.

L'option **Visible/Caché** permet de faire apparaître ou disparaître la figure selon l'état de la variable Booléenne désignée. Ex : Ici si *Voir* est égal à True alors le texte sera visible sur le synoptique sinon il sera caché.

Texte sur plusieurs lignes

Insérez le caractère **#** dans la chaîne de caractères, afin que le système passe à la ligne suivante. Le caractère **#** est dans tous les cas remplacé par un espace.

Exemple :

ARSOFT#International Affichage sous visual I/O :

☞ **Note** : L'objet groupe n'a aucune fonctionnalité, ce n'est qu'un rectangle incluant un texte.
Il a pour fonction d'entourer d'autres objets. Il est impératif de toujours tracer le groupe avant les objets qui seront entourés. Dans le cas contraire Visual I/O détectera le clic souris sur le groupe et jamais sur les objets à l'intérieur de celui-ci.

Programmation sur texte et groupe

Dans le cas ou l'expert n'est pas utilisé, vous pouvez changer le texte par la procédure SetTextMessage.

Changement de texte au passage de la souris.

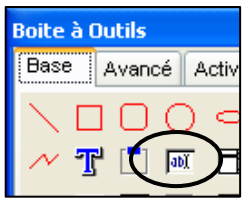
```
// *** Text1 ************************************
SObject Procedure Text1;
Begin
 If Is_me Then SetTextMEssage('Bonjour')
          Else Settextmessage('Au Revoir');
End;
```

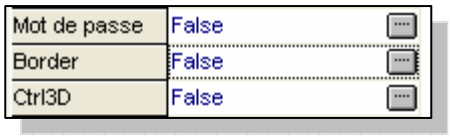
Changement de fonte au passage de la souris.

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

NB : **SetText** et **GetText** fonctionnent également et peuvent remplacer **SetTextMessage** et **GetTextMessage**.

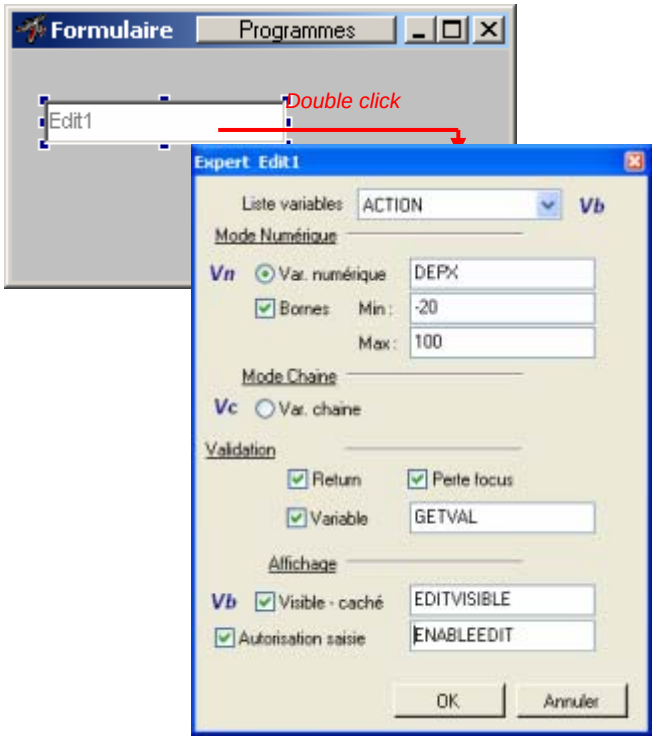
Champ d'édition – EditBox





Frappez <F6> pour afficher la boite de propriétés
Border à false permet de faire disparaître le contour de la zone de saisie .Ctrl3D permet de modifier l'aspect du contour.
Mot de passe permet de remplacer les caractères par le caractère *.

Acquisition d'une valeur numérique



Depx : reçoit la valeur numérique frappée par l'opérateur
Min : Valeur frappée minimum autorisée.
Max : Valeur frappée maximum autorisée.

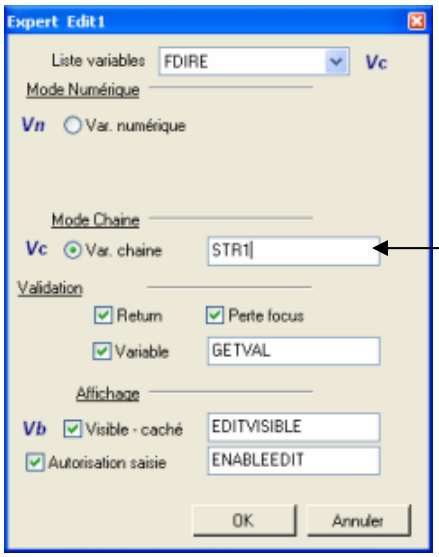
La variable Depx reçoit la valeur frappée si :
L'opérateur frappe la touche <**Entrée**>
Ou
Le champ de saisie est quitté par la touche <**tab**>
Ou
la souris est déplacée dans un autre champ de saisie.
Ou
La variable GETVAL passe à True.

☞ **Note** : Validation peut être une des conditions ou une combinaison de ces 3 conditions.

Autoriser la saisie dans un champ de saisie

Visible – Caché : Si cette variable est false le champ de saisie est caché dans le cas contraire le champ de saisie est visible.
Autorisation saisie : Si cette variable est true la saisie est autorisée (les caractères frappés sont affichés dans le champ de saisie) sinon les caractères frappés par l'opérateur sont ignorés.

Acquisition d'une chaîne de caractères



La variable chaîne STR1, suit les mêmes règles que l'acquisition d'une variable numérique.
Les caractères frappés par l'opérateur seront envoyés dans cette variable de type String.

Si aucune option de validation n'est cochée, chaque caractère frappé est directement entré dans la chaîne STR1.

Programmation sur les EditBox.

Changement de couleur de fond quand le champ de saisie à le curseur

```
SObject Procedure Edit1;
Begin
 If Get_Focus Then SetFillcolor($80FFFF)//* Si edit box a le focus coloriage en jaune
              Else FirstColors;    //* Si pas le focus revient aux couleurs d'origine
End;
```

Tester la frappe de la touche <Entrée>

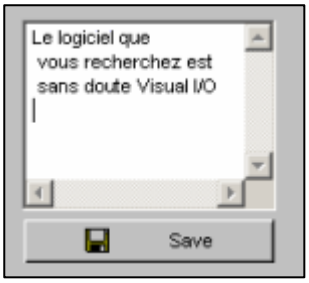
Tester la saisie d'une valeur numérique

```
// *** Edit2 *************
SObject Procedure Edit2;
Var
  Status : Integer;
  R      : Real;
Begin
  If EditCR then // *** test la frappe de la touche <Entrée>
  Begin
    Val(GetText,R,Status);    // *** GetText récupère le texte frappé
    If Status<>0 Then Clear; // **** Val converti la valeur en reel si status=0.
  End;
  If EditEsc Then SetText('0'); // *** Test la frappe de la touché <Echap> alors raz
End;
```

Passer au champ de saisie suivant

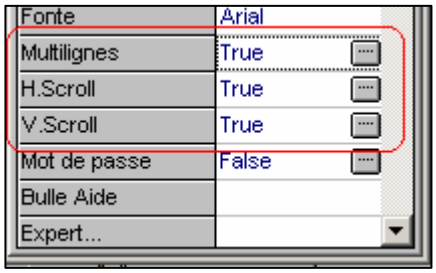
```
Uses Windows ;  // *** Windows.vpu Contient les déclarations VK_F1, VK_F6 etc..
// *** Si <F6> on passe au champ suivant cette ligne peut être inscrite n'importe ou dans
// ** le synoptique.
If Getvirkey=VK_F6 Then NextFocus;
```

Sauvegarder un champ de saisie dans un fichier de type texte



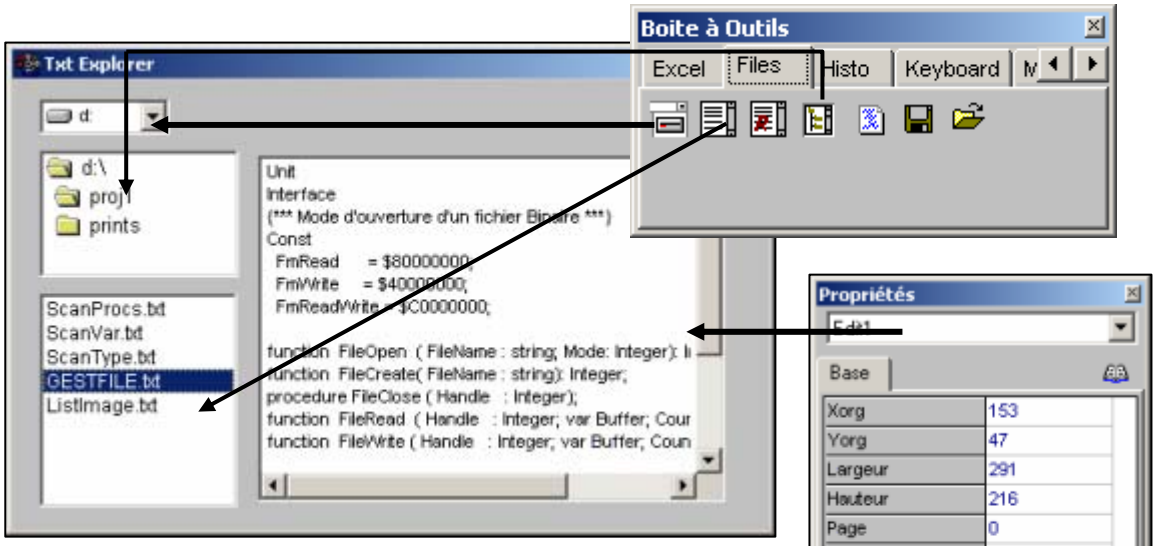
Le champ de saisie est multi lignes, autorisant ainsi la saisie sur plusieurs lignes.

Des scrollers verticaux et horizontaux sont possibles en les spécifiant dans la boite de propriétés.

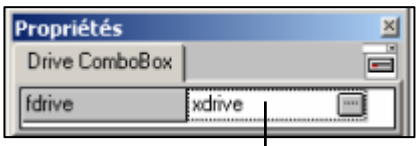



```
Uses GestFile ; // *** pour utilisation de la fonction DirectoryCreate
// *** Button2 ************************************
SObject Procedure Button2;
Var
  Fic : Integer;
  I   : Integer;
  S   : String;
Begin
  If Button Then   // *** Click sur bouton Save ?
  Begin
    Commut_Fig('Edit4'); // *** Passe sur le champ de saisie multiLignes
    If DirectoryExist('D:\Tempx\Doc')=False Then // *** Si le répertoire n'existe pas
        DirectoryCreate('D:\Tempx\Doc');   // *** On le crée
   FileTextAppend(Fic,'D:\tempx\Doc\Readme.txt'); // *** Ouverture du fichier ou création
    For I:=0 To GetNumLines-1 Do
    Begin
      GetLine(S,I);   // *** Récupération de chaque ligne
      Writeln(Fic,S); // *** écriture à la fin du fichier ouvert
    End;
    FileTextClose(Fic); // *** Fermeture du fichier
  End;
End;
```

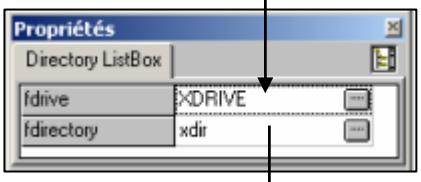
Visualiser un fichier texte dans un champ de saisie



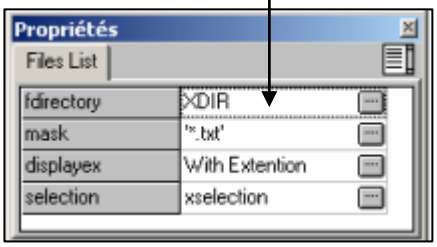
Composant **Drive Combobox**



Composant **Directory ListBox**



Composant **Files List**




```
// *** Edit1 ************************************
SObject Procedure Edit1;
Var
 Fic : Integer;
 TS  : String;
Begin
  // *** Test si le contenu de la chaîne à changée et non vide
 If StrChange(XSelection) and (XSelection<>'') Then
 Begin
   Clear;                          // *** Efface le champ de saisie
   FileTextOpen(Fic,XSelection);  // ** Ouverture du fichier
   hidecontrol;           // ** Cache le champ de saisie (pour être plus rapide)
   While Not(Eof(Fic)) Do         // ** balaie tout le fichier
   Begin
     Readln(Fic,TS);    // ** Lit ligne par ligne
     Addstring(TS);     // *** Ajoute la ligne lue
   End;
   Showcontrol;           // *** réaffiche le champ de saisie et son contenu
   FileTextClose(Fic);  // *** Ferme le fichier texte
 End;
End;
```

Note :

Cette fonction existe toute faite dans la librairie ***Bonus32.vpu*** *– Function* ***ListBoxAddFile****;*
Les instructions *HideControl* et *Showcontrol* permet de cacher momentanément le champ de saisie le temps de le remplir de lignes lues dans le fichier. Ces 2 instructions peuvent être éliminées, dans ce cas l'opérateur verra le champ de saisie se remplir ligne à ligne. Ce code peut être aussi utilisé derrière une **listbox** normale. Le résultat sera identique sauf que l'opérateur ne pourra pas modifier les lignes. Comme les autres objets le champ de saisie accepte le coloriage avec les procédures SetFillColor et SetColor pour la couleur du texte.

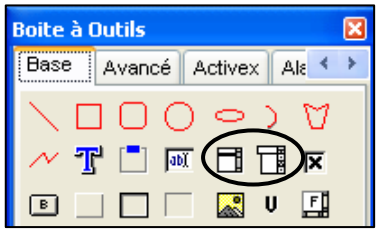
Drag & drop d'une zone de saisie vers une autre.

```
// *** Edit2 ************************************
SObject Procedure Edit2;
Begin
  Drag(GetText); // ** Drag permet de récupérer le texte renvoyé par GetText
End;
// *** Edit3 ************************************
SObject Procedure Edit3;
Begin
 If Drop Then  // ** Drop indique si l'opérateur relache le drag & drop dans l'objet
   Settext(Getdrop); // ** Si oui forcage de l'editbox avec le texte de Drag
End;
```

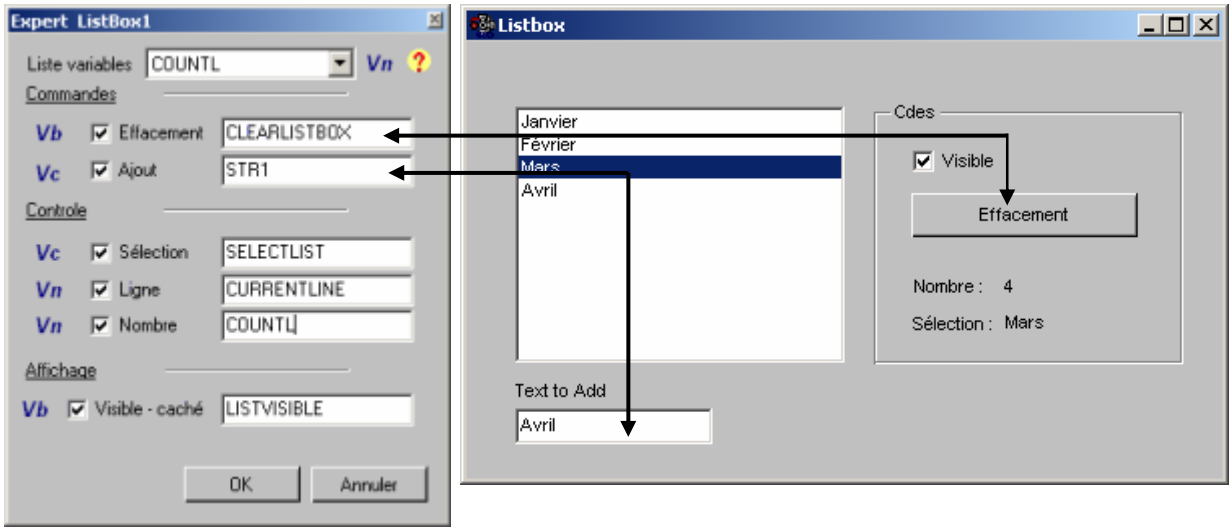
Sélectionne le contenu d'une zone de saisie.

```
// *** Edit3 ************************************
SObject Procedure Edit3;
Begin
  If B1 Then SetSelection(0,255); // ** Sélection du premier caractère au 255ème
End;
```

Liste déroulantes – ListBox, ComboBox



Expert



Effacement : Permet d'effacer le contenu de la Listbox quand cette variable est à True.

Ajout : Est une variable de type Chaîne de caractères, qui lorsque son contenu change est ajouté à la suite des chaînes existantes dans la Listbox ou s'intercale si la Listbox en triée (voir propriétés).

Sélection : Est une variable de type Chaîne de caractères, qui reçoit le contenu de la ligne pointée (Barre vidéo inverse dans la ListBox).

Ligne : Est une variable numérique recevant le numéro de ligne pointée (Barre vidéo inverse dans la ListBox). La première ligne porte le numéro 0. Si aucune ligne n'est présente dans la ListBox cette variable reçoit la valeur -1

Nombre : Est une variable numérique recevant le nombre de ligne dans la ListBox.

L'option **Visible/Caché** permet de faire apparaître ou disparaître la figure selon l'état de la variable Booléenne désignée. Ex : Ici si *ListVisible* est égal à True alors la ListBox sera visible sur le synoptique sinon elle sera cachée.

Le ComboBox a un fonctionnement similaire à la ListBox.

Cas d'une listbox ou d'un combobox personnalisé

Il est possible d'afficher des bitmaps, textes en couleur et d'aligner les textes différemment. Passer la listbox en traçage personnalisée.
Frappez **<F6>** puis fixez l'attribut Personnalisé à True.

La procédure Addstring permettant d'ajouter des lignes à la listbox fonctionne bien évidemment comme avant mais aussi avec des fonctionnalités supplémentaires.

Addstring('**Bitmap**'+#1+'**Texte**'+#1+'**Couleur**'+#1+<'**Décalage en pixel 'facultatif**'>.

Programmation sur les Listbox.

Initialiser le contenu d'une listbox ou d'un Combobox.

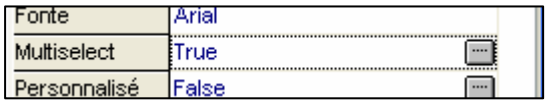
```
IF Start_Visu Then //** Test du lancement de l'application
Begin
  Addstring('Visual IO'); //** Ajoute du Texte Visual IO
  Addstring('Visual PLC); //** Ajoute du Texte Visual PLC
End;
```

Envoyer le contenu d'une listebox vers une liaison série.

```
Uses Serial;
// *** ListBox1 ************************************
SObject Procedure ListBox1;
Var
 I : Integer;
Begin
  If Go Then
  Begin
    Opencom('1','9600','E','8','1'); // ** Ouvre le port série Com1
    For I:=0 to Getcount-1 Do        // ** Balaie toutes les lignes
    Begin
      Comwrite(1,Getstring(I)); // ** Envoie la ligne sur COM1
    End;
    Disablecom(1); // ** Ferme le port de communication COM1
  End;
End;
```

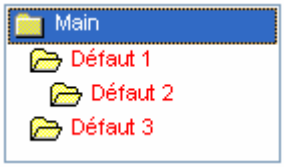
Envoyer les lignes sélectionnées dans une listebox vers un fichier.

Frappez **<F6>** dans la boite de propriétés, vous devez basculer l'attribut **Multiselect** à **true**




```
// *** ListBox1 ************************************
SObject Procedure ListBox1;
Var
 Fic : Integer;
 I   : Integer;
Begin
  If Go Then
  Begin
    Filedelete('Temp.txt');             // ** Efface le fichier final
    Filetextappend( Fic,'Temp.txt');    // ** ouvre le fichier en ajout
    For I:=0 to Getcount-1 do           // ** balaie toute la listebox
    Begin
      If Getselected(I) then            // ** si la ligne est sélectionnée
         Writeln(Fic,Getstring(I));     // ** on l'envoie dans le fichier
    End;
    Filetextclose(Fic);    // ** Ferme le fichier
  End;
End;
```

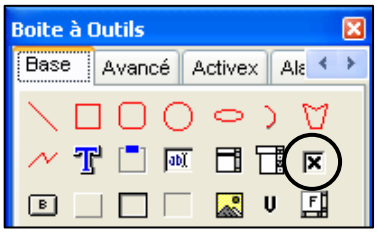
Ajout de chaînes de caractères dans une listebox personnalisée.

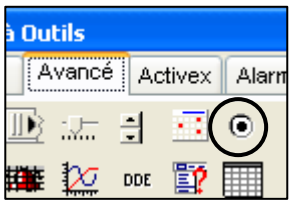


Affichage de bitmap et décalage.
Les fonctions ci-dessus fonctionnent également en mode personnalisé.
Note : La function **VisuBMP**(Num ligne) renvoi le nom du bitmap au début de La ligne

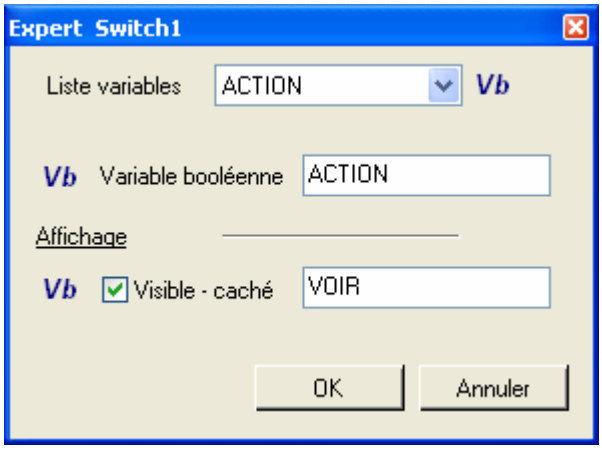
```
// *** ListBox1 ************************************
SObject Procedure ListBox1;
Begin
  If Go Then
  Begin
   Addstring('VPCLOSEDFOLDER'+#1+'Main'+#1+'$FF0000'+#1);     // Aucun décalage
   Addstring('OpenFolder'+#1+'Défaut 1'+#1+'$0000FF'+#1+'10'); //décalage de 10 pixels
   Addstring('OpenFolder'+#1+'Défaut 2'+#1+'$0000FF'+#1+'20'); //décalage de 20 pixels
    Addstring('OpenFolder'+#1+'Défaut 3'+#1+'$0000FF'+#1+'10');
  End;
End;
```

Case à cocher et Radiobouton – Checkbox, Radiobutton





La case à cocher et le radio bouton permet d'affecter constamment un état à une variable booléenne.



◄ **Variable booléenne** reçoit l'état de la case à cocher. Si celle-ci est cochée la variable booléenne est forcée à True sinon le contraire.

L'option **Visible/Caché** permet de faire apparaître ou disparaître la Case à cocher selon l'état de la variable Booléenne désignée.

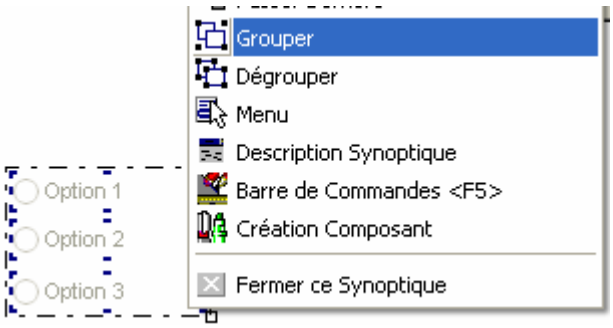
Ex : Ici si *Voir* est égal à True alors la case à cocher sera visible sur le synoptique sinon elle sera cachée.

☞ **Note** : La case à cocher ne fonctionne quand commande, autrement dit elle ne peut pas visualiser un état changeant. Utilisez la case à cocher en commande exclusivement et non en visualisation.

Radiobouton

Le radio Bouton permet aussi d'affecter un état binaire (0 ou 1) à une variable booléenne.
Un Radio-Bouton si cliqué passe dans l'état coché et ne peut être décoché que par un clic sur un autre radio bouton.

Pour que des Radio Boutons est une exclusion mutuelle, il faut les grouper dans l'éditeur de Visual I/O.
Pour ce faire, sélectionnez les Radio Boutons devant avoir une exclusion mutuelle puis groupez-les par la commande grouper du menu Editer.
Grâce à cette méthode il est possible de définir des groupes de Radio Boutons.



◄ Ici le fait de grouper les Radio boutons *Option 1,2 et 3* ensemble, ceux-ci travailleront en exclusion mutuelle.

Programmation sur les cases à cocher et les radio boutons.

Tester si une case à cocher a changée d'état

```
// *** Switch5 ************************************
SObject Procedure Switch5;
Begin
  If ISmodified Then A5:=A5+1; //** Si l'état à change, Ismodified passé à true
                               //  sur un tour de cycle
End;
```

Cocher et décocher une case à cocher ou un radio bouton par programme

```
// *** Switch5 ************************************
// ** Fonctionne aussi bien sur un radio bouton ou une case à cocher
SObject Procedure Switch5;
Begin
  If Go Then Check  //* Si la variable Go=True  alors on coche la case ou le radio
        Else Uncheck; // ** Si la variable Go=False alors on Décoche la case
End;
```

☞ **Note** : Si ce programme est appliqué à un radio bouton le fait de le coché décoche automatiquement les autres radio boutons puisque ceux ci on une exclusion mutuelle. Concernant le radio bouton la procédure **SetRadio** (**State** : Boolean)**;** peut être utilisée à la place de check et uncheck

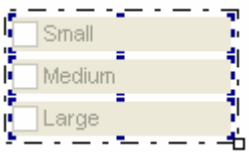
Tester l'état d'une case à cocher

```
// *** Switch5 ************************************
SObject Procedure Switch5;
Begin
  If (GetCheck=1) And (IsModified) Then A5:=A5+1; //** L'état à change et que la
                  // case est cochée alors on incrémente le mot A5
End;
```

Tester l'état d'un Radio bouton

```
// *** Switch5 ************************************
SObject Procedure Switch5;
Begin
  If (GetRadio) And (IsModified) Then A5:=A5+1; // * L'état à change et que la
                            //case est cochée alors on incrémente le mot A5
End;
```

Remise à zéro de toutes les cases à cocher dans un groupe

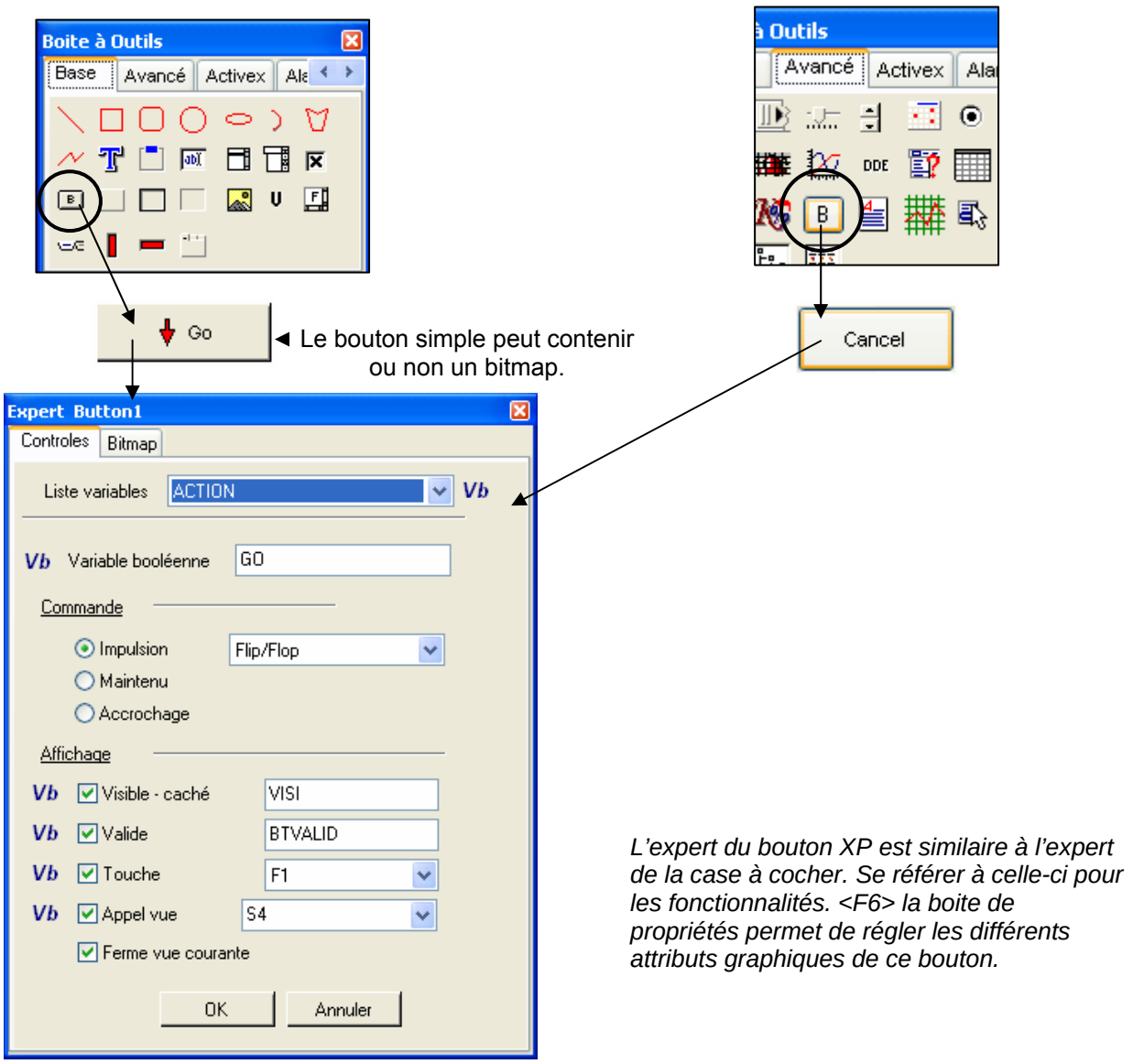


◄ Les cases à cocher doivent être groupées.
Si la case à cocher *Small* est la première tracée (Tête du groupe) le programme ci-dessous doit être inscrit derrière.

```
// *** Switch5 ************************************
SObject Procedure Switch5;
Begin
  If Go Then
  Begin
    Repeat          // ** démarre sur Small
      Uncheck;      // ** décoche la case à cocher en cours
    Until Next_Group=False;  // ** passe à la figure suivante ou renvoi false
  End;                       // ** si c'est la dernière
End;
```

Bouton

Deux types de boutons sont disponibles :
Le bouton simple dans l'onglet Base qui est gérer entièrement par le système Visual I/O
Le bouton au look XP qui est dans l'onglet Avancé et qui est géré par Windows XP



L'expert du bouton XP est similaire à l'expert de la case à cocher. Se référer à celle-ci pour les fonctionnalités. <F6> la boite de propriétés permet de régler les différents attributs graphiques de ce bouton.

La **variable booléenne** reçoit le résultat de l'action sur le bouton.

Le bouton peut être :

- **impulsion** :
- **Flip/Flop** : Inverse à chaque clic l'état de la variable booléenne.
- **Pulse** : Passe à un sur un tour de programme
- **Reset** : Remet la variable booléenne à False (0).
- **Set** : Met la variable Booléenne à True (1).

Maintenu : Type de bouton poussoir ou la variable booléenne reste à True (1) tant que le bouton est enfoncé.
Accrochage : Sur clic de l'opérateur le bouton s'enfonce et reste enfoncé passant la variable booléenne à True (1) tant que celui-ci est enfoncé. Un clic successif de l'opérateur relâche le bouton et passe la variable booléenne à False (0).

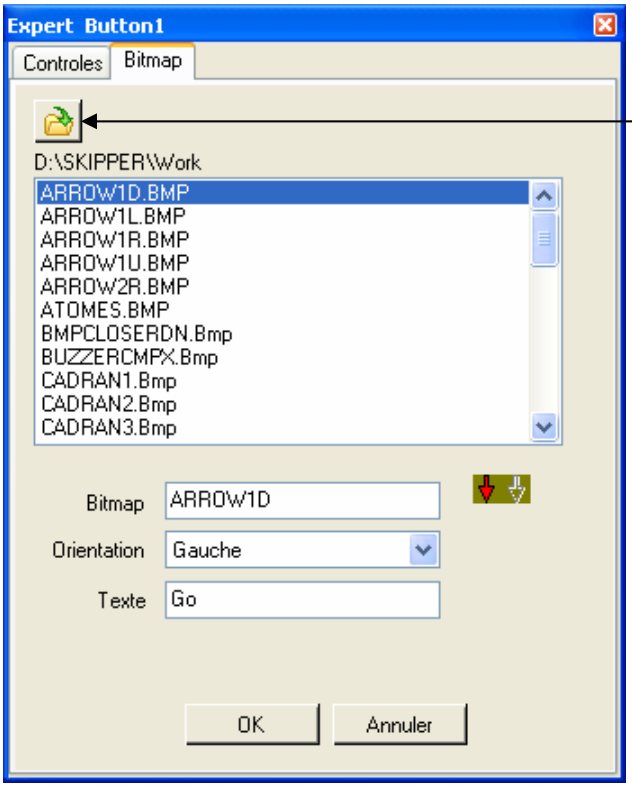
Visible/Caché permet de faire apparaître ou disparaître le Bouton selon l'état de la variable Booléenne désignée.

Valide : Permet de valider ou dévalider (Bouton grisé) le bouton selon l'état d'une variable booléenne. Si cette variable est à False le bouton est grisé ainsi que le Bitmap éventuel un clic de la souris n'a aucun effet sur celui-ci.

Touche : Permet d'affecter une touche de Fonction ou alphanumérique à ce bouton valide si un booléen est spécifié dans le champ Variable booléenne.

Appel Vue : Permet sur clic du bouton d'appeler un synoptique.

Fermeture Vue courante : Permet de fermer le synoptique contenant ce bouton.
Si le synoptique est le principal (Premier affiché au lancement de l'application) l'application complète est fermée.



Bitmap : Permet d'afficher un petit Bitmap à coté du texte du bouton.

Pour rechercher un Bitmap, la navigation est possible par le bouton.

Le bitmap choisi est alors recopié dans le projet courant.

Le Bitmap doit être constitué de 2 parties. La partie représentant le Bitmap Valide et une autre partie le Bitmap inactif (Grisé). Ceci correspondant aux 2 états possibles du bouton. Un Bitmap 32x16 est une dimension correcte.
Une liste de boutons est disponible dans la sous-directorie d'installation Buttons.

Orientation du Bitmap : Permet de positionner le Bitmap à Gauche, Droite, Haut, Bas.

☞ **Note** :

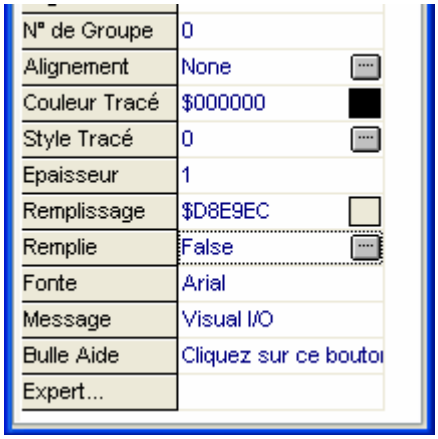
Texte du bouton sur deux lignes

Si le bouton est trop peu large pour afficher le texte, celui-ci peut être inscrit sur plusieurs lignes la deuxième ligne commence au premier espace trouvé.

◄ Ici le bouton est assez large pour afficher le texte sur sa longueur

◄ Ici le bouton n'est pas assez large pour afficher le texte sur sa longueur. Celui-ci est coupé au niveau du caractère

Boite de propriétés



Remplissage permet de fixer la couleur du bouton.
Couleur tracé permet de fixer la couleur du texte du bouton.

Remplie =False permet de créer un bouton plat sans contour.
Le contour apparaissant lorsque l'opérateur passe la souris dessus.

Bulle d'aide : permet comme tout les objets de visual I/O d'avoir une petite fenêtre d'aide qui apparaît quand on passe la souris sur le

Programmation sur les boutons

Attention la fonction Button n'est valide que si l'expert ne contient aucun Booléen !

Tester si un bouton est cliqué

Si cliqué le texte du bouton est changé par le texte *Stop*

```
// *** Button1 ************************************
SObject Procedure Button1;
Begin
 If Button then SetText('STOP'); //** La fonction button passé à vrai sur 1 tour de cycle
End;
```

Changer la couleur d'un bouton en fonction d'une valeur numérique

Les couleurs ClRed.. sont définies dans Windows.vpu d'ou la présence de **Uses Windows.**

```
Uses Windows;
// *** Button1 ************************************
SObject Procedure Button1;
Begin
 Case A1 of
      1 : Setfillcolor(clRed); // ** Setfillcolor colorie l'intérieur du bouton
      2 : Setfillcolor(clBlue);
      3 : Setfillcolor(clGreen);
      4 : Setfillcolor(clOlive);
      5 : Setfillcolor(clPurple);
      Else Firstcolors; // ** Revient aux couleur d'origine si A1<>12,3,4 ou 5
 End;
 SetColor(CLWhite XOR GetFillColor); // Pour que le texte soit toujours visible
End;
```

Changer la couleur d'un bouton quand la souris arrive sur celui-ci

```
// *** Button1 ************************************
SObject Procedure Button1;
Begin
  If Arrive Then Setfillcolor(ClRed); // Quand la souris Arrive colorie en rouge
  If Depart Then Firstcolors;    // Quand la souris sort du bouton revient aux
End;                             // couleurs d'origine
```

Cacher un bouton en fonction d'un état booléen

Permet de mettre en service ou non des fonctionnalités en fonctions des utilisateurs ou conditions externes.
Cet fonctionnalité est aussi disponible en automatique dans l'expert bouton.

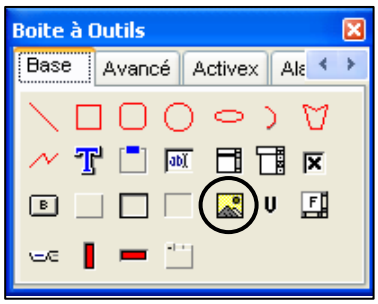
```
// *** Button1 ************************************
SObject Procedure Button1;
Begin
 if Action Then Showcontrol
           Else Hidecontrol;
End;
```

Changer la fonte et le style d'un bouton

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

☞ **Note** : Les Lignes, Cercles, Rectangle, support, Ellipse, Polygon, Polylines peuvent aussi servir de bouton par l'attribut *détection clic* de l'expert.
La function **Button** fonctionne aussi **sur toutes les figures graphiques**.

Image



L'objet Image permet d'afficher des images aux formats :

Bitmap	: .BMP
Méta fichier	: .WMF
Jpeg	: .JPG

Ces images sont incluses dans l'exécutable final, simplifiant ainsi l'installation de vos applications.

Permet d'afficher une image selon les conditions choisies. L'expert est très proche de celui du texte.

2 types de traitements sont possibles :

Le test d'une variable booléenne avec ses 2 conditions **Vrai ou Faux**.
Selon l'état, l'image paramétrée sera affichée. Si la case à cocher (dessous cercle clignotant) est validée, l'image clignotera quand la condition sera vérifiée.

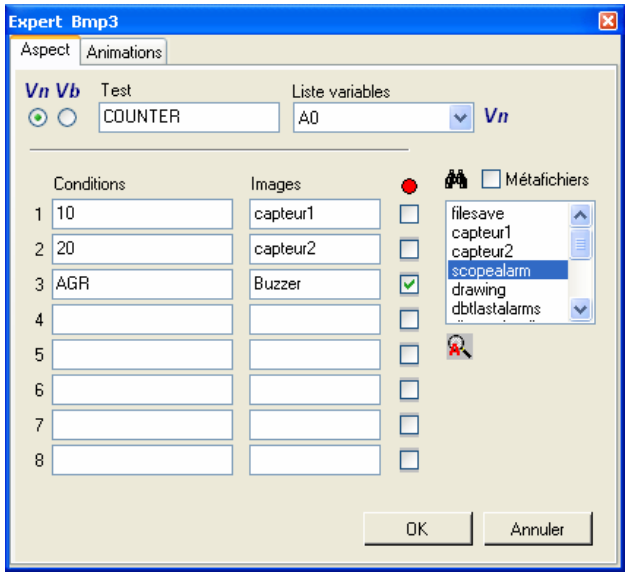
Amenez la variable désirée du combobox *Liste Variables* vers le champ de saisie *Test*. ou frappez le nom de la variable à tester.
Si cette variable n'existe pas, le système la créée automatiquement en la rajoutant dans la liste des variables globales.
La liste des images du projet en cours est affichée dans la ListBox de droite.
L'extension des Images est .WMF ou .BMP

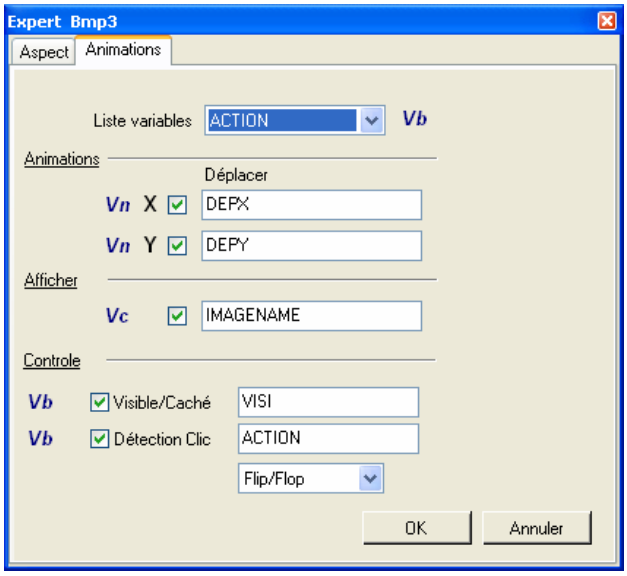
L'autre type de test est une comparaison numérique. ►

La variable testée n'est plus une variable booléenne mais une variable numérique (Word, Integer ou Real).

8 Conditions sont disponibles.

A chaque condition un nom d'image est affectée avec ou sans clignotement.





En fonction du champ de saisie, la liste des variables du ComboBox 'Variables' change. Pour la rubrique Animations seul les variables numériques sont autorisées.

Dans le champ Déplacer en X ou Y indiquez la variable contenant le déplacement relatif aux coordonnées d'origine.

Ex : Si la variable DepX contient 10, alors l'image se déplacera de 10 Pixels en X par rapport à sa position d'origine que vous aviez tracé sous l'éditeur.

DepX : Indique le déplacement absolu par rapport à la position d'origine (en design).
DepY : Indique le déplacement absolu par rapport à la position d'origine (en design).

Afficher **: Permet d'afficher le nom d'une image contenu dans une variable de type Chaîne de caractères.**

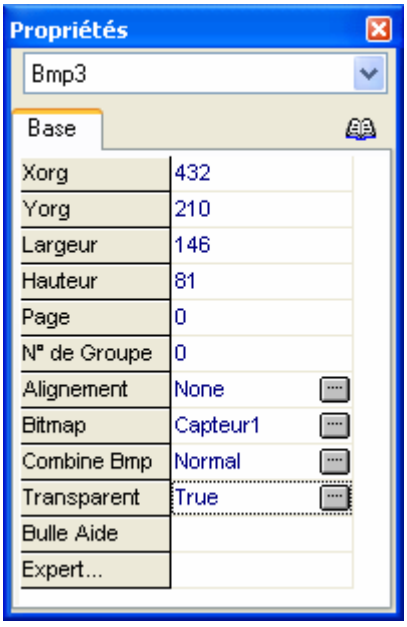
Visible/Caché permet de faire apparaître ou disparaître l'image selon l'état de la variable Booléenne désignée. Ex : Ici si *Voir* est égal à True alors l'image sera visible sur le synoptique sinon elle sera cachée.

Détection clic permet d'associer à une variable booléenne le clic de la souris sur l'image. Ici si l'opérateur clic sur l'image alors la variable *Action* sera mise à True.
Plusieurs types d'actions sont possibles par sélection dans le ComboBox 'Type'.

Flip/Flop : Inverser la variable booléenne à chaque Clic de la souris sur l'image.
Set : Mise à True (1) de la variable booléenne à chaque Clic de la souris sur l'image.
Reset : Mise à False (0) de la variable booléenne à chaque Clic de la souris sur l'image.

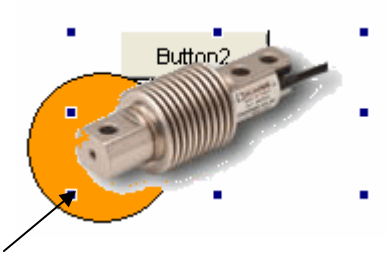
☞ **Note** : La variable chaîne de caractère spécifiée dans le champ affiche reçoit en run le nom de l'image à afficher. Ce nom peut contenir l'extension ou non. Dans le deuxième cas Visual I/O cherche sur le disque dans le répertoire courant le nom avec extension.BMP sinon cherche avec extension .wmf ou jpg. Si visual I/O ne trouve pas, la recherche se fait dans les ressources de l'exécutable crée.

Bitmap transparent



L'attribut **transparent** permet de rendre le bitmap affiché transparent. La couleur transparente est déterminée par la couleur du pixel en bas à gauche du bitmap.

Combine BMP ne fonctionne que si Transparent est false et à pour fonction de combiner les pixels du bitmap avec ceux du fond donnant ainsi des effets graphiques.



La couleur de transparence est la prise de valeur du pixel en bas à gauche de l'image Bitmap affichée.

Programmation sur les images

Change l'image affichée en function d'une valeur numérique

```
// *** Bmp3 ************************************
SObject Procedure Bmp3;
Var
  ImageBase : String; // ** Cette variable appartient à l'objet Bmp3
Begin
 If Start_Visu Then ImageBase:=GetTextMessage; //  met en mémoire le nom de
                                               // l'image de départ
 Case A1 of
      1 : SetTextMessage('Capteur1'); // ** Affiche Capteur1.Bmp
      2 : SetTextMessage('Capteur2'); // ** Affiche Capteur2.Bmp
      3 : SetTextMessage('Capteur3');
      4 : SetTextMessage('Capteur4');
      Else SetTextMessage(ImageBase); // ** revient à l'image de départ
 End;
End;
```

☞ **Note** :La variable ImageBase comme éventuellement d'autres variables est sauvegardée et appartient à l'objet BMP3.

Déplace sur l'écran l'image à l'aide de la souris

```
// *** Bmp3 ************************************
SObject Procedure Bmp3;
Begin
   MoveFig; // ** cet instruction permet de déplacer à la souris n'importe quel
               objet
End;
```

Ajuste le bitmap au cadre image

StretchBmp ne fonctionne pas quand l'attribut transparent=true.

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

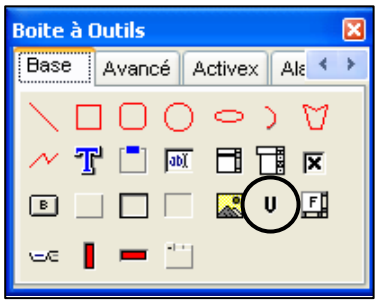
Transfert d'une image bitmap dans une autre par son handle

```
// *** Bmp3 ************************************
SObject Procedure Bmp3;
Var
 HD : Integer;
Begin
if action then
 Begin
   HD:=Gethandlebmp;    // ** récupère le Handle du bitmap BMP3
   Commut_fig('BMP4');  // ** passé sur le bitmap BMP4
   Sethandlebmp(HD);    // ** Copie le handle BMP3 dans le handle BMP4
 End;
End;
```

☞ **Note** : Grâce au handle du bitmap il est possible de récupérer toutes les informations le concernant (taille, planes, bitsPixel Etc..) à l'aide de la fonction Windows **GetObject**. Cette programmation système sort du cadre de Visual I/O voir SDK windows.

Pour exemple voir Source Function **DuplicateBitmap** dans **Graphics.PAS**.

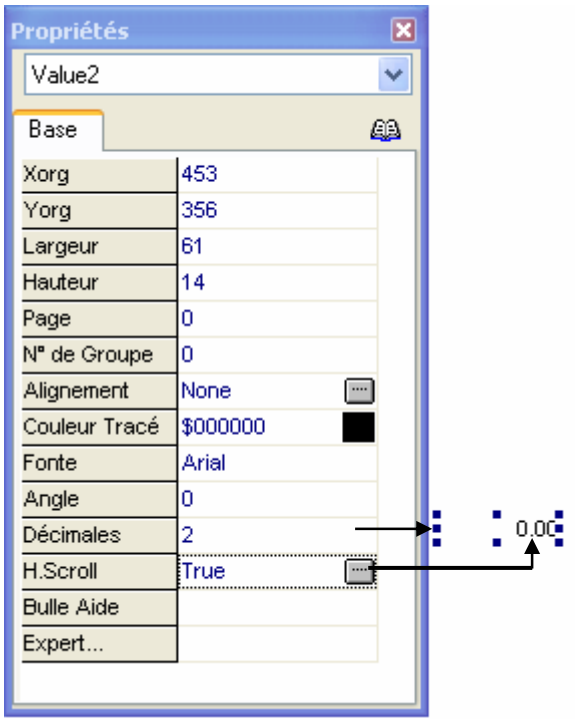
Valeur numérique

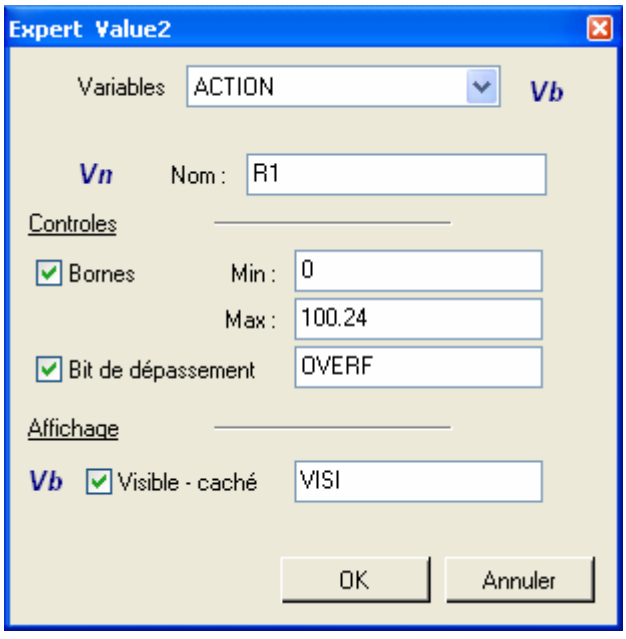


Pour régler le nombre de décimales affichées, appelez la boite propriétés en frappant la touche <**F6**>. puis réglez la propriété **Decimales.**

Pour centrer le texte à droite réglez la propriété **H.scroll à true**

Angle permet d'afficher la valeur sous un angle donnée





La variable numérique à afficher est contenue dans le champ de saisie Nom.

Cette variable peut être bornée. C'est à dire qu'elle ne pourra pas sortir des **bornes** indiquées.
Si ce cas se produit, le bit de dépassement indiqué passera à True et la valeur numérique sera automatiquement limitée aux bornes.

Le bit de dépassement passe automatiquement à True mais doit être remis à 0 (False) par programme.

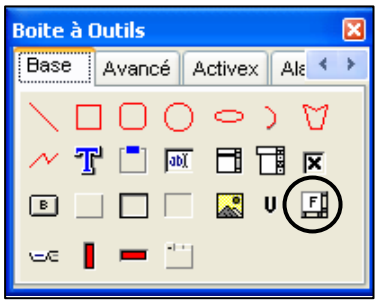
Visible-Caché : permet d'afficher ou de cacher la valeur numérique en fonction de l'état de la variable booléenne indiquée.

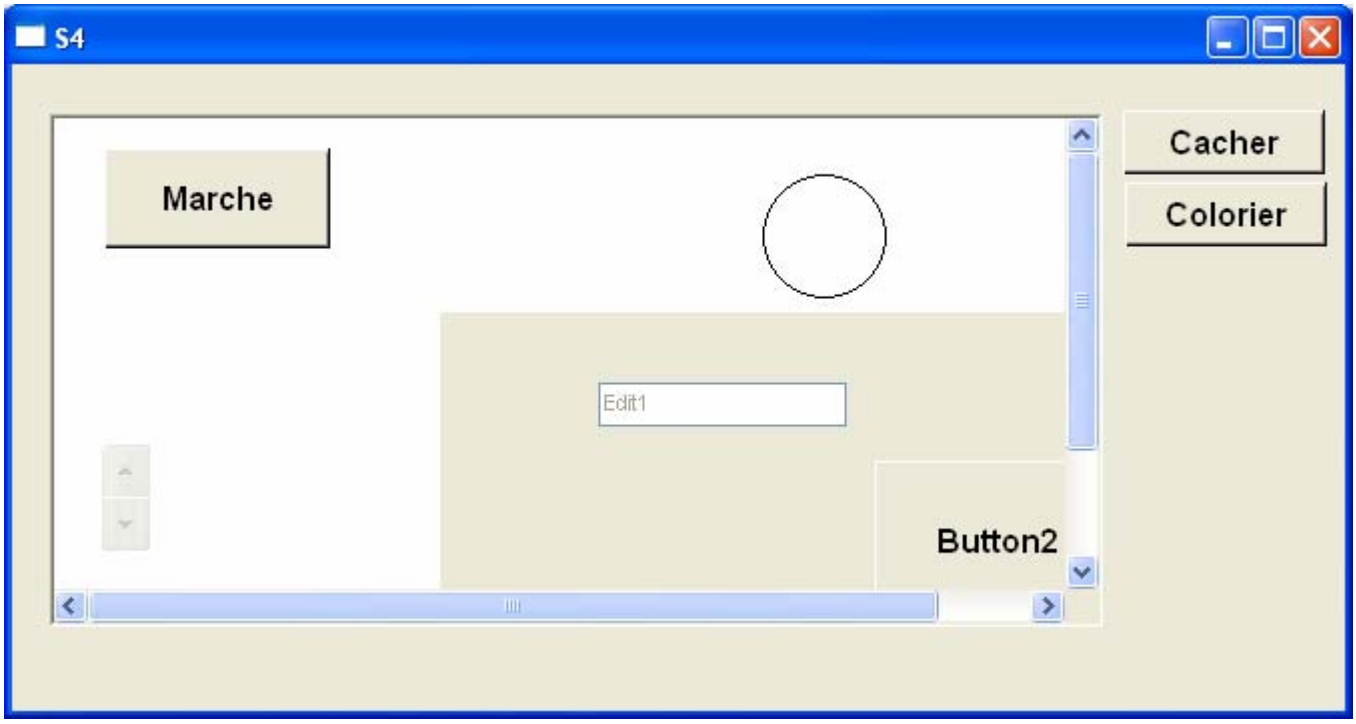
Programmation sur les valeurs numériques

Ajuste le nombre de décimales d'une valeur numérique et fixe sa valeur

```
// *** Value2 ************************************
SObject Procedure Value2;
Begin
  If action then
  Begin
    Setndecimales(5);       // *** Fixe 5 chiffres après la virgule
    SetValue(Getvalue+1);   // *** Incrémente de 1 la valeur Courante.
  End;
End;
```

Fenêtre interne





La fenêtre interne ne dispose pas d'expert. Elle est un container permettant d'afficher des figures.
Si toutes les figures ne sont pas affichables (par manque de place par exemple), des barres de Scroll apparaissent automatiquement pour permettre la visualisation de toute la surface de la fenêtre.

Il est possible alors de faire apparaître ou disparaître la fenêtre par programme.
Sélectionnez la fenêtre puis par clic droit de la souris appelez l'éditeur de programme :

Programmation sur les fenêtres internes

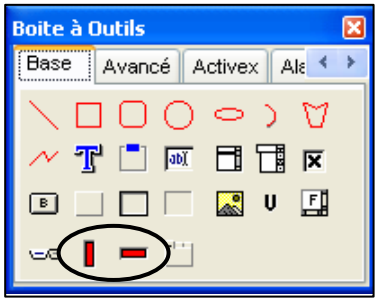
Afficher,cacher et colorier le fond d'une fenêtre interne

```
Uses Windows;
// *** Window1 ************************************
SObject Procedure Window1;
Begin
 if action then Showcontrol   // ** Affiche la fenêtre
           Else Hidecontrol;  // ** Cache la fenêtre
  If B0 then Setfillcolor(ClRed) // ** Colorie en rouge le fond
        Else Firstcolors;        // ** revient aux coleurs originales
End;
```

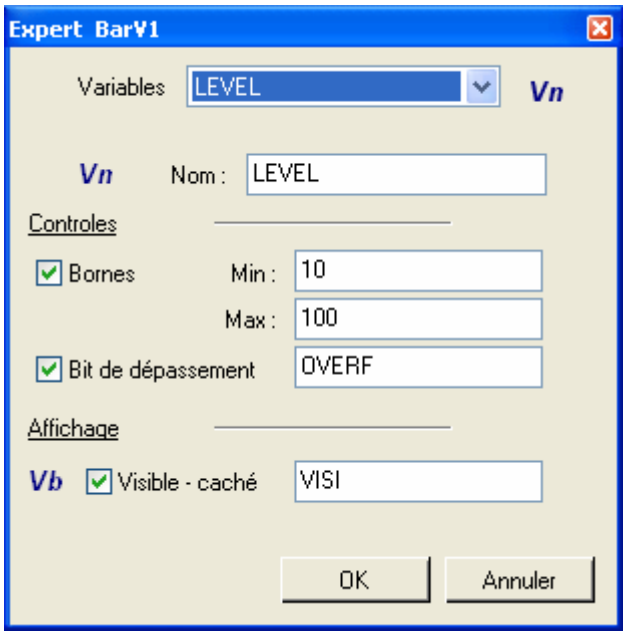
☞ **Note** :

L'instruction **ShowControl** affiche n'importe quelle figure graphique.
L'instruction **HideControl** Cache n'importe quelle figure graphique.

Bargraph Vertical et horizontal



Le bargraph Vertical et horizontal, suit les mêmes règles que celles de la valeur numérique. Ces 2 bargraphs évoluent de base de 0 à 100 pour leur hauteur Mini et Maxi mais ils peuvent être mis à une autre échelle en remplissant les champs Min et Max.



La variable **numérique** à afficher sous forme de Bargraph est contenue dans le champ de saisie **Nom**.

Cette variable peut être bornée. C'est à dire qu'elle ne pourra pas sortir des **bornes** indiquées.

Si ce cas se produit, le bit de **dépassement** indiqué passera à True et la valeur numérique sera automatiquement limitée aux bornes.

Ces bornes indiquent aussi la plage d'évolution du Bargraph. (Ici Le bargraph sera à 0 lorsque la variable Niveau sera Inférieure ou égale à 10 .
Le bargraph sera maximum lorsque la variable Niveau sera supérieur ou égal à 100).
Le **Bit** de **dépassement** passe automatiquement à True mais doit être **remis à 0** (False) **par programme.**

Sens d'évolution des bargraph

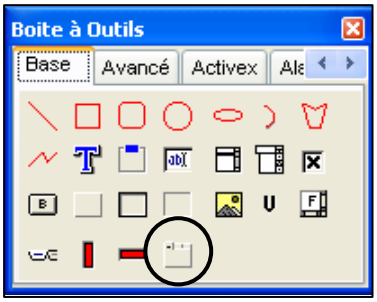
Le sens d'évolution du bargraph vertical et le sens de tracé au design. Si vous tracez le bargraph de haut en bas celui-ci évoluera de haut en bas (0 à 100%). Si vous le tracez de Bas en Haut celui-ci évoluera de bas en haut.
Le sens d'évolution du bargraph horizontal et le sens de tracé au design. Si vous tracez le bargraph de droite vers la gauche celui-ci évoluera de la droite vers la gauche. Si vous le tracez de la gauche vers la droite, celui-ci évoluera de la gauche vers la droite.

Programmation sur les bargraph

Fixer la valeur du bargraph

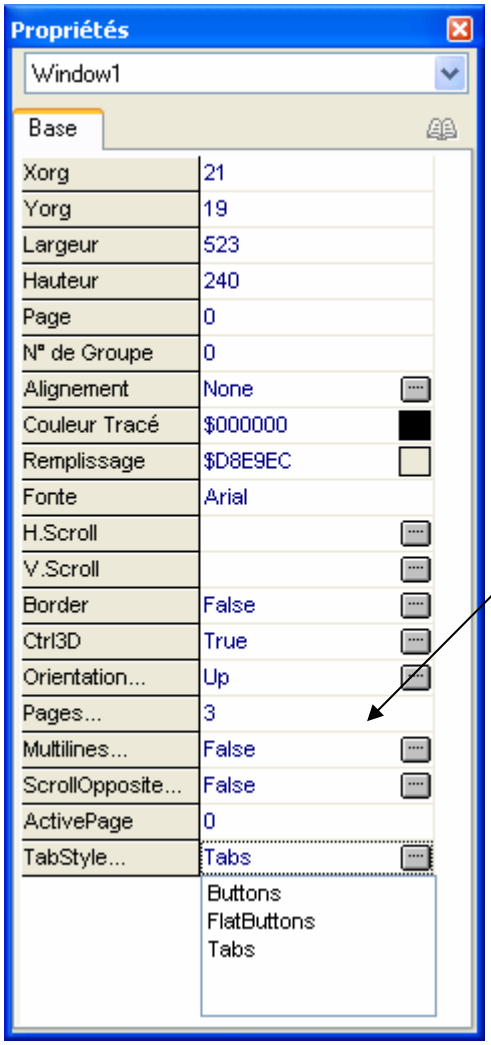
```
// *** BarV1 ************************************
SObject Procedure BarV1;
Begin
  Setvalue(A0); // ** Fixe la valeur du bargraph directement
End;
```

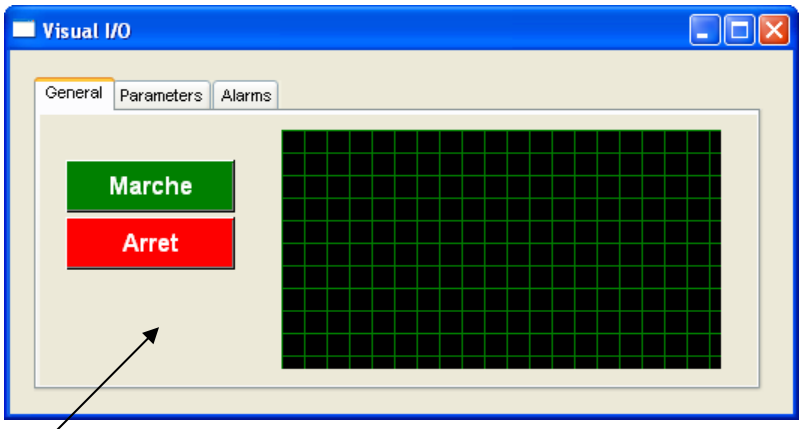
Page Control - Classeur

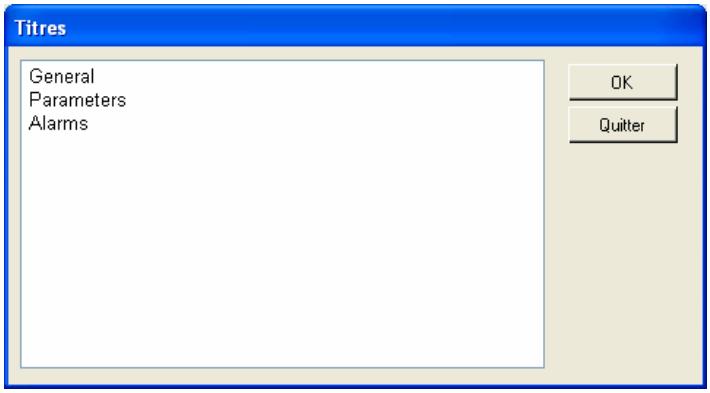


L'objet Page Control, ne dispose pas d'expert. Il permet simplement de créer un dossier contenant des onglets.
Ces onglets peuvent bien sur contenir toutes les figures graphiques de Visual I/O.
Ce contrôle permet de définir plusieurs pages logiques ou des sections d'informations à l'intérieur de la même fenêtre.

Le Page Control ce paramètre sous forme d'onglets à l'aide de la boite de propriétés en le sélectionnant puis en frappant la touche **<F6>**.







Double cliquez sur la cellule en face du texte Pages..
Une boite de dialogue apparaît permettant de définir les intitulés de chaque page

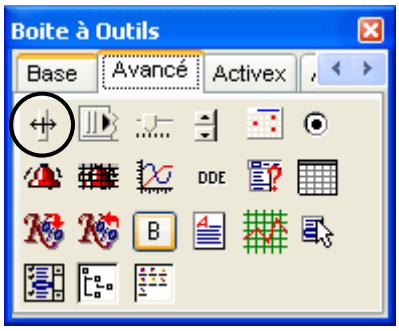
Différents type d'onglets sont proposés dans **Tabstyle**. Le style de base est Tabs (comme représenté ci-dessus). **Orientation** permet aux onglets d'être représentés au dessus ou au dessous.
ScrollOpposite permet lorsque le page control n'est pas assez large de passer une partie des onglets en dessous.

Programmation sur les page control

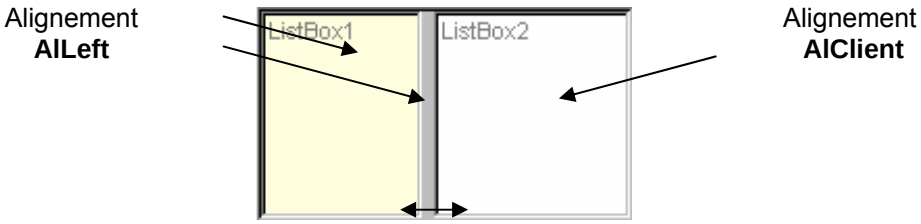
```
// *** Button3 ************************************
SObject Procedure Button3;
Begin
 if Button Then
 Begin
   Commut_fig('Window1');
   Setpage(Getpage+1); // *** passe à la page suivante
 End;
End;
```

☞ **Note** : La première page porte le numéro 0.

Splitter - Séparateur



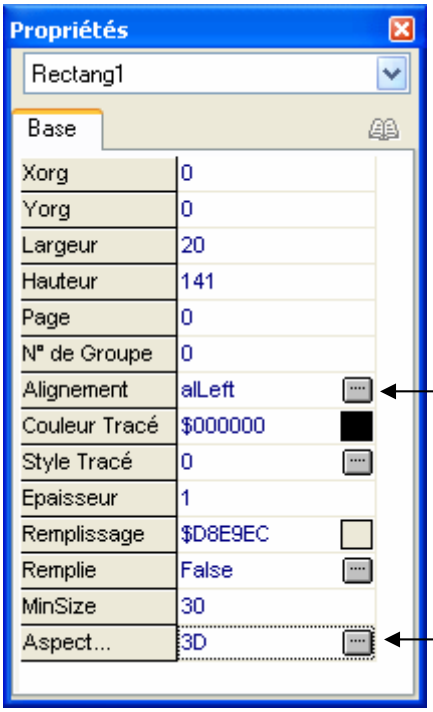
Ajoute dans un synoptique un séparateur entre 2 contrôles alignés pour permettre aux utilisateurs de redimensionner ces contrôles lors de l'exécution. Le séparateur se situe entre un contrôle aligné sur l'un des bords de la fenêtre et les contrôles remplissant le reste de la zone client.
Alignez le séparateur de la même façon que le contrôle ancré sur le bord de la fiche.
Quand l'utilisateur déplace le séparateur, il redimensionne le contrôle ancré. Cela change la zone client de la fiche, et les contrôles remplissant le reste de la zone client sont redimensionnés en conséquence.



Définissez au design la propriété **<F6>:**

MinSize Pour spécifier la taille minimum que doit respecter le diviseur lorsque son contrôle voisin est redimensionné.

Aspect Pour donner ou non au bord du diviseur un aspect biseauté 3D.



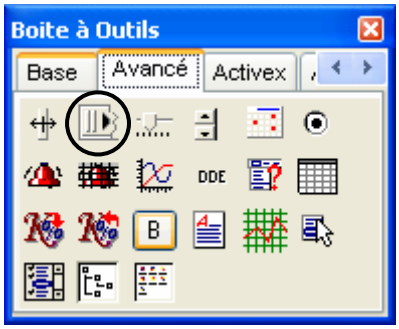
Les propriétés du splitter.

Le nom du contrôle apparaît comme un rectangle avec des propriétés supplémentaires.

Alignement Alleft : viens en buté sur le contrôle de gauche listbox1 qui lui est déjà aligné à gauche.
Minsize : permet de fixer la taille minimum du contrôle à gauche du splitter (ici ListBox1).

Aspect : 3D ou non pour la barre du splitter.

Coolbar – Barre d'outils



Barre multiple.

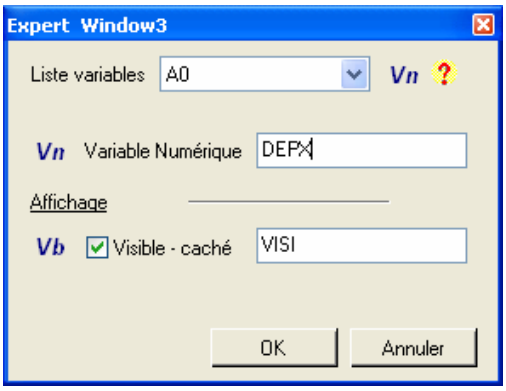
Affiche une collection de contrôles fenêtrés à l'intérieur de bandes pouvant être déplacées ou redimensionnées.
L'utilisateur positionne les contrôles en faisant glisser la poignée de dimensionnement qui se trouve à gauche de chaque bande.

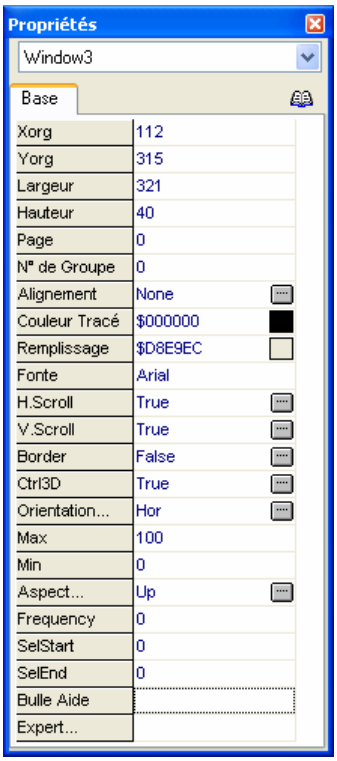
Les bandes de CoolBar reçoivent uniquement des contrôles Windows.
Pour ajouter un contrôle Windows dans une bande choisissez-le dans la boîte à outils de Visual I/O puis tracez-le dans la barre souhaitée. Le contrôle Windows s'adapte automatiquement à la largeur et la hauteur de la bande (Cas ci-dessus de Edit1).
Si vous souhaitez placer plusieurs contrôles dans une bande, tracez d'abord une fenêtre vierge (Window de la boîte à outils) celle-ci s'adapte aussi à la taille de la bande. Coloriez-la de la couleur grise en contour et en remplissage (boîte de propriétés de Visual Pascal) basculez ses propriétés H-Scroll et V-Scroll à false.
Vous pouvez alors tracer tout élément dans cette fenêtre Windows gérée par la bande.
Pour ajouter une nouvelle Bande : Agrandissez le cadre général de CoolBar puis amenez un nouveau contrôle Windows dans la zone agrandie. Une nouvelle barre se dessine alors automatiquement.
AutoSize=True de la boîte de propriété permet de retailler correctement le pied de CoolBar en s'alignant automatiquement sur le bas de la dernière bande.

Trackbar - Potentiomètre

L'expert permet d'affecter une variable numérique au TrackBar.

L'option Visible-Caché permet d'affecter une variable booléenne permettant selon son état d'afficher ou de cacher celui-ci.





Les cellules Max et Min permet de régler les valeurs Mini et Maxi qui seront affectées à la variable numérique spécifiée dans la boite de dialogue expert ci dessus.
Aspect : permet de changer la forme du curseur.

Frequency : permet d'afficher des graduations sur le curseur du trackbar.

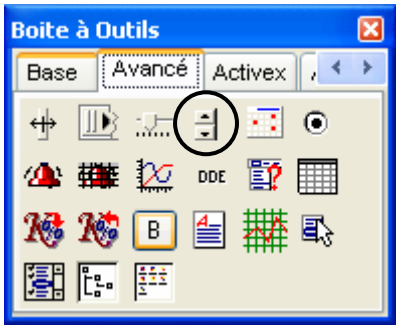
Orientation : permet de définir si le trackbar est horizontal ou vertical.

Programmation sur les trackbars

GetTrackBar et SetTrackbar permettent de fixer et de retourner la valeur courante d'un trackbar.

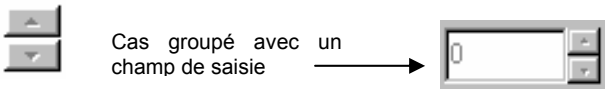
Le trackbar permet aussi d'afficher deux marques : une marque minimum et l'autre maximum en dessous le trackbar et en plus des graduations. Ceci par les procédures **SetTrackStart**(value);
SetTrackEnd(Value);
C'est 2 procédures de limitent pas le trackbar, ceci est à faire par programme (comparaison de position avec max et limitation par SetTrackBar (Valuemax) par exemple).

Updown – Flèches haut/bas

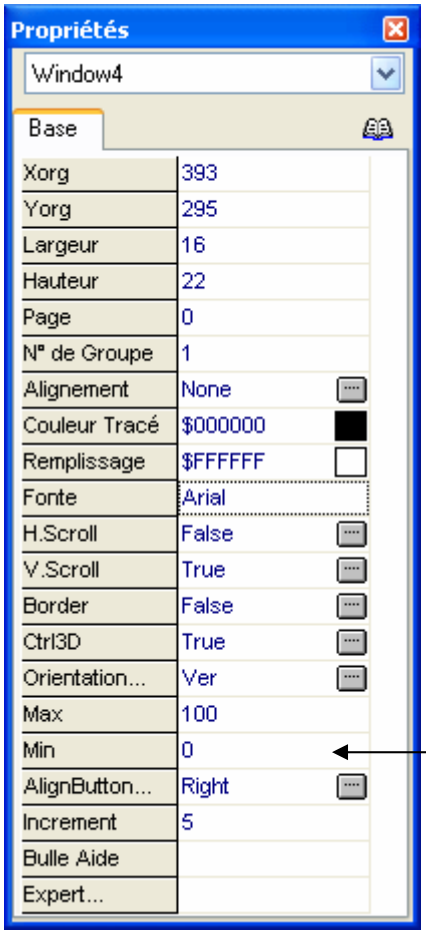


Flèches Haut/Bas. Boutons flèches haut et bas permettant d'incrémenter et de décrémenter des valeurs.

Ce contrôle groupé avec un champ de saisie

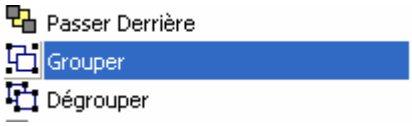


Quand l'utilisateur clique sur la boîte texte ou sur les boutons, la focalisation est attribuée au texte du contrôle. L'utilisateur peut entrer directement une valeur dans le contrôle ou utiliser les boutons pour incrémenter ou décrémenter la valeur.



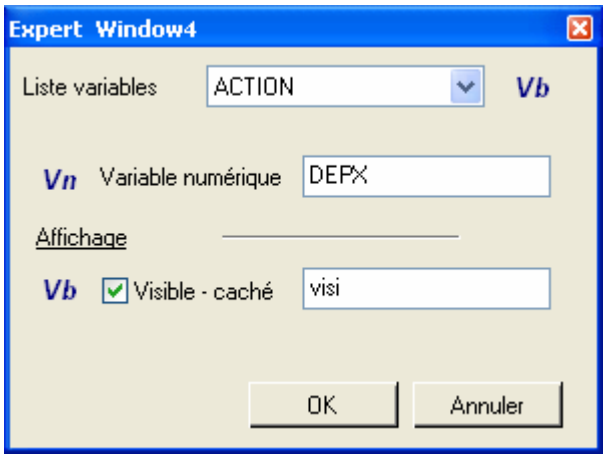
La valeur d'incrémentation est fixée au design dans la boîte de propriétés par la propriété **Increment**.

Pour les grouper sélectionnez les 2 éléments puis appliquez la commande grouper du menu flottant (Clic droit de la souris).



AlignButton de la boite de propriétés, permet de placer le contrôle UpDown à droite ou à gauche du champ de saisie.

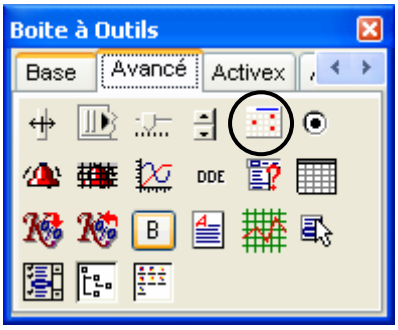
Les Maxi et Mini applicables à la valeur numérique spécifiée dans l'expert sont réglables dans la boite de propriétés dans les cellules **Max et Min**.



La valeur numérique à incrémenter est spécifiée dans le champ Variable Numérique.

L'option Visible – Caché permet d'afficher ou de cacher le composant UpDown en fonction de l'état d'une variable booléenne.

DateTime Picker - Sélecteur Date/Heure

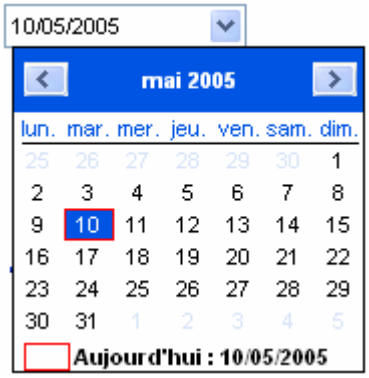


Affiche une boîte liste permettant de sélectionner des dates ou des heures.

Les utilisateurs peuvent sélectionner une date dans le calendrier, ou bien sélectionner la date ou l'heure en les faisant défiler à l'aide des flèches ou en les tapant.

Vous devez avoir la dernière version de COMCTL32.DLL, généralement située dans le répertoire Windows\System ou

Aspect final



Programmation sur le Datetime picker

Récupérer la date sélectionnée

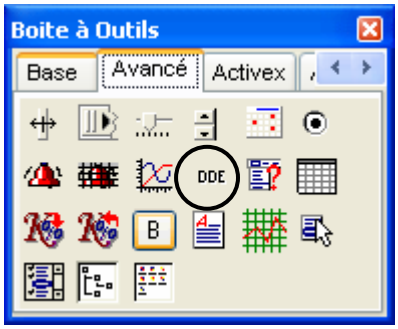
```
// *** Window5 ************************************
SObject Procedure Window5;
Begin
  Ladate:=GetText; // ** Récupère la date affichée
End;
```

Ajoute trois jours à la date sélectionnée

```
// *** Window5 ************************************
SObject Procedure Window5;
Var
 R,R1 : Real;
Begin
  LaDate:=GetText;            // ** Récupère la date affichée
  R :=StrtoDateTime(Ladate);  // ** Converti
  R1:=StrToTime('12:00:00');  // *** 12 Heures
  R1:=(R1*2)*3;               // ** 24Heures * 3 jours
  R:=R+R1;                    // ** Additionne 3 jours à la date de base
  LaDateBis:=DateTimeToStr(R);     // *** Date + 3 jours en chaîne
End ;
```

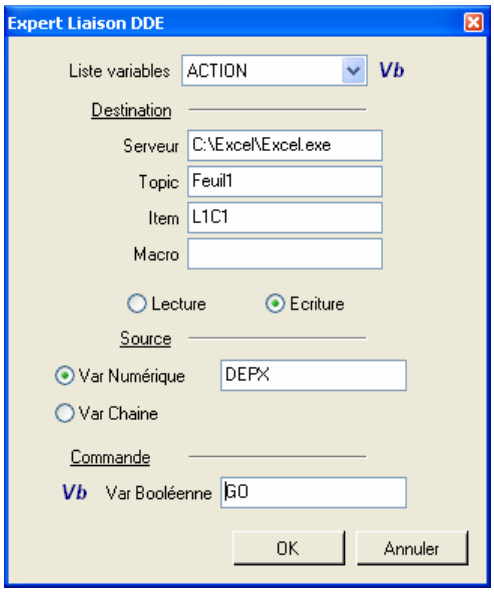
StrToDateTime : Converti une chaîne de caractères contenant une date et une heure en numérique
StrToTime : Converti une chaîne de caractères contenant une heure en numérique ;
DateTimeToStr : Converti un numérique représentant une date et heure en chaîne de caractères.

DDE Client



La connexion DDE permet l'envoi ou la lecture de variable dans une application dite serveur DDE (ex : Excel).

Le DDE permet à deux applications de communiquer entre elles. Les applications doivent être lancées. L'une des applications est alors serveur tandis que les autres sont clients. Le serveur peut aussi être le client d'autres applications.
Mécanisme plus ou moins obsolète aujourd'hui ! gardé ici pour compatibilité.



Serveur Nom du serveur DDE.
Topic Conversation
Item Elément de conversation

Macro Macro commande à envoyer vers le serveur DDE.

Lecture ou écriture dans le serveur.

Source :
Var Numérique Variable numérique à envoyer dans le serveur.
Var Chaine Variable chaîne à envoyer dans le serveur.

Commande :
Var Booléenne : Variable booléenne si à vrai valide la conversation DDE vers le serveur.

☞ **Note** En indiquant le chemin complet du serveur DDE, Visual I/O ouvre ce serveur lors de la première communication si cette application n'est pas déjà lancée.

Cependant si l'application Serveur DDE n'est pas Lancé, la première communication est perdue.
Si vous désirez lancer une communication avec plusieurs éléments (Item) en cible, vous pouvez utiliser les instructions :

OpenDDE, CloseDDE, DDEGetData, DDESendData.

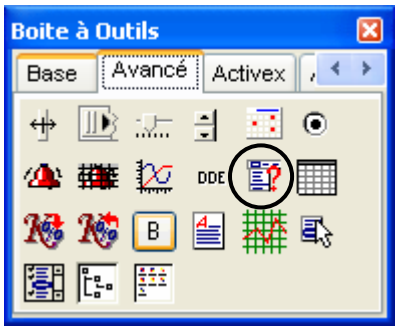
Programmation avec le mécanisme DDE

Rempli les 10 premières colonnes dans une fiche excel

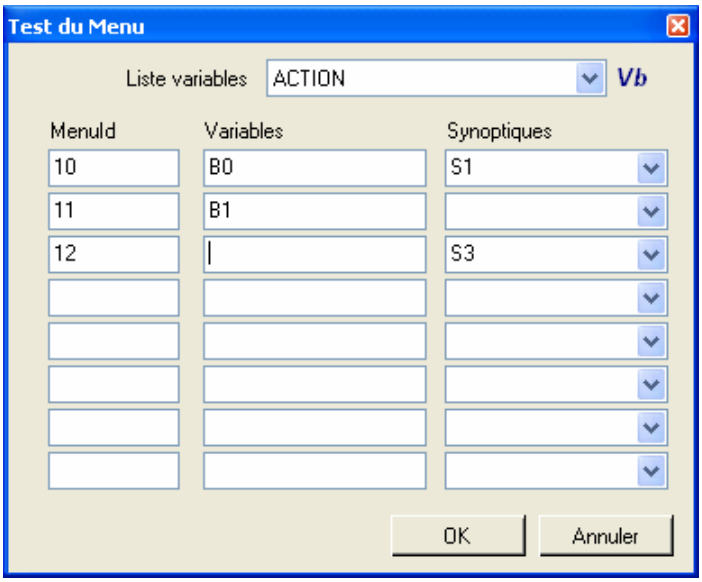
```
// *** Bmp2 ************************************
SObject Procedure Bmp2;
Begin
  IF Go Then
  Begin
    OpenDDE('C:\Excel\Excel.exe'); // ** Ouvre Excel
    For I :=1 To 10 do
       DDESendData('Feuil1','L1C'+IntToStr(I),IntToStr(I)); //* Envoie vers excel
    CloseDDE;
  End;
End;
```

☞ **Note** : Un **serveur DDE** pour les applications Visual I/O et Visual PLC est disponible gratuitement sur le site ARSOFT (ww.arsoft-i nt.com).

Test Menu principal



Cet expert permet de faire une comparaison avec les identificateurs de menu et l'identificateur de menu sélectionner.
Il permet de mettre à True une variable ou d'appeler un synoptique donné.



Dans la colonne **MenuId** indiquez les identificateurs de rubrique de menu.
Dans la colonne **Variables** indiquez les variables à mettre à true lorsque l'opérateur clique sur la rubrique d'un menu ayant l'identificateur mentionné dans la colonne Menuid.
Dans la colonne **Synoptiques**, indiquez le nom du synoptique à afficher lorsque l'opérateur clique sur la rubrique d'un menu ayant l'identificateur mentionné dans la colonne Menuid.

☞ **Note** : Voir le chapitre Générer un Menu principal. Au lieu de tester les textes des rubriques de menu, Visual I/O affecte un numéro (Id) unique à chaque rubrique.
Ceci facilite le test, sachant que le numéro ID est stable et le programmeur peut changer les intitulés sans avoir à retoucher le programme.
Fonctionne aussi bien sur avec les identificateurs des menus flottants.

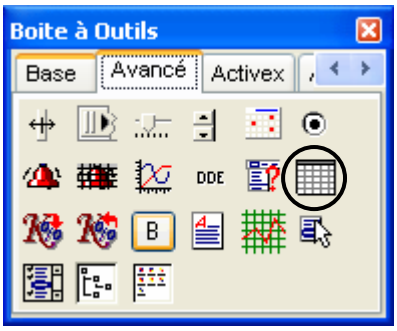
Programmation test du menu

Tester la sélection d'un item du menu

```
// *** Button3 ************************************
SObject Procedure Value1;
Begin
 If WinMenu='MAIN' Then  // ** Si le menu clique est dans le syno main
 Case Menuid of
   10 : Display('Parameters');
   11 : B0:=True;
   12 : B0:=False;
   13 : CloseAppli;
 End;
End;
```

☞ **Note** : *La variable **WinMenu** renvoie le nom du synoptique dans lequel le menu à été cliqué.*
*La variable **MenuStr** renvoie l'intitulé de l'item cliqué dans le menu.*

Grille de données

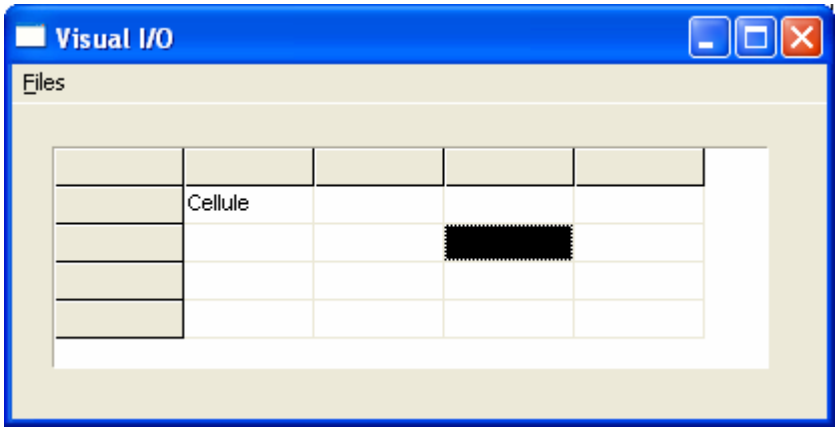


Une grille regroupe un ensemble de données (valeurs ou autres) dans un tableau.
Visual I/O met à disposition du programmeur un ensemble de fonctions permettant de manipuler par programme les données de la grille.
Une grille peut contenir du texte, des bitmaps et des contrôles Windows (Listebox, ComboBox, Cases à cocher, palette de couleur), des lignes de différentes couleurs.

Visual I/O intègre un ensemble des fonctions permettant de gérer les grilles.

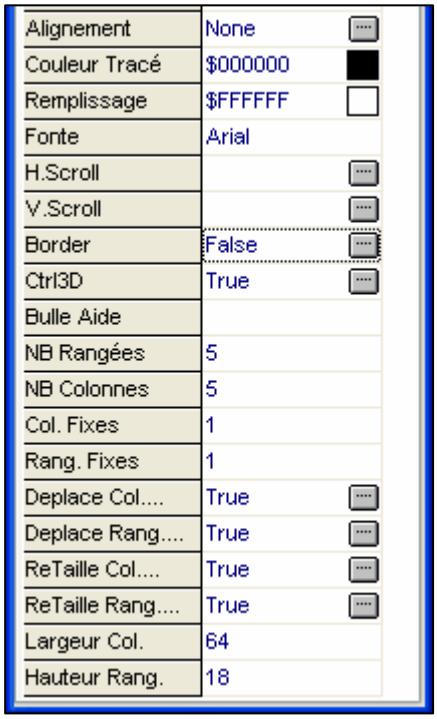
Cette grille à 5 rangées fixes et 5 colonnes fixes,symbolisées par des cellules grisées dans lesquelles aucune saisie n'est possible.

Celles-ci sont toujours localisées à gauche pour les colonnes (à partir de la colonne 0) et en haut pour les rangées (à partir de la rangée 0).



Les grilles permettent dans une application Visual I/0 d'afficher ou de saisir des textes dans des cellules.
Les cellules sont organisées en **Rangées** et **Colonnes.** La première colonne et la première rangée sont numérotées 0 .
Plus de 1 million de **Cellules** peuvent êtres créés.

Frappez **<F6>** pour afficher les propriétés de la grille :



Border et Ctrl3D permettent de modifier le contour de la grille.

NB rangées : Indique le nombre de rangées dans la grille.
NB colonnes : Indique le nombre de colonnes dans la grille.

Col Fixes : Indique le nombre de colonnes à gauche de la grille qui ne défilent pas. (grisées).
Rang. Fixes : Indique le nombre de lignes en haut de la grille qui ne défilent pas. (grisées).

Deplace Col : Indique si l'opérateur aura la possibilité de déplacer à l'aide de la souris les colonnes en cliquant sur une colonne fixe et la faire glisser sur une autre. (intervertir 2 colonnes).
Deplace Rang : Indique si l'opérateur aura la possibilité de déplacer à l'aide de la souris les rangées en cliquant sur une rangée fixe et la faire glisser sur une autre. (intervertir 2 rangées).
Retaille Col : Indique si l'opérateur aura la possibilité de retailler les colonnes en prenant à l'aide de la souris entre 2 colonnes.
Retaille Col : Les colonnes mobiles peuvent être redimensionnées individuellement.
Retaille Rang. : Les lignes mobiles peuvent être redimensionnées individuellement.
Largeur Col. : Indique la largeur des colonnes à l'initialisation de la grille.
Hauteur Rang. Indique la hauteur des rangées à l'initialisation de la grille.

Programmation sur les grilles

Afficher un texte en rouge dans les cellules fixes

```
// *** Window1 ************************************
SObject Procedure Window1;
Begin
  If go Then
  Begin
   GSetTextFixedColor(255); // Fixe la couleur d'écriture dans les cellules fixes
   GsetText(1,0,'Title1'); // *** Ecriture dans Colonne 1
   GsetText(2,0,'Title2'); // *** Ecriture dans Colonne 2
   GsetText(3,0,'Title3'); // *** Ecriture dans Colonne 3
   GsetText(4,0,'Title4'); // *** Ecriture dans Colonne 4
  End;
End;
```

Sauvegarde le contenu d'une grille dans un fichier texte

```
// *** Window1 ************************************
SObject Procedure Window1;
Var
 Fic,X,Y : Integer;
 TS : String;
Begin
  If Go then
  Begin
    FileDelete('Temp.TXT');   // ** Efface le fichier final
    FileTextAppend(Fic,'Temp.TXT'); // ** Ouvre le fichier en ajout
    For X:=0 To GGetColCount-1 Do  // ** Scan toutes les colonnes
     For Y:=0 to Ggetrowcount-1 do // ** Scan toutes les rangées
     Begin
       TS:=GGetText(X,Y); // ** Récupère le contenu de la cellule
       Writeln(Fic,TS);   // ** écrit la valeur dans la chaine
     End;
    FileTextClose(Fic); // ** Ferme le fichier en final
  End;
End;
```

Charge le contenu d'un fichier texte dans une grille

```
// *** Button2 ************************************
SObject Procedure Button2;
Var
 Fic,X,Y : Integer;
 TS : String;
Begin
  If LoadGrid and (FileExists('Temp.TXT')) then // ** Si fichier existe
  Begin
    Gcleargrid; // ** Efface toute la grille
    FileTextOpen(Fic,'Temp.TXT'); // ** Ouvre le fichier en lecture
    For X:=0 To GGetColCount-1 Do  // ** Scan toutes les colonnes
     For Y:=0 to Ggetrowcount-1 do  // ** Scan toutes les rangées
     Begin
       Readln(Fic,TS);   // *** Lit une ligne dans le fichier texte
       GSetText(X,Y,TS); // ** ecrit dans la cellule spécifiée
     End;
    FileTextClose(Fic); // ** Ferme le fichier en final
  End;
End;
```

☞ **Note** : Le programme sauvegarde le contenu ci-dessus est écrit derrière la grille concernée. Le programme Charge le contenu d'un fichier est écrit derrière le bouton.. Ceci est possible si le synoptique ne contient qu'une seule grille. Alors toutes les instructions concernant la grille sont dirigées vers l'unique grille. Dans le cas de plusieurs grilles, il est nécessaire de faire un **Commut_Fig ou Go2** avant afin de spécifier la grille de destination.

Détecter si il y a eu un changement dans une des cellules de la grille et la colorie en rouge

```
// *** Window1 ************************************
SObject Procedure Window1;
Var
 X,Y : Integer;
Begin
  If Gcellchange Then  // ** Détecte si il y a eu un changement dans une cellule
  Begin
    X:=Ggetcolchanged;  // ** récupère la colonne modifiée
    Y:=Ggetrowchanged;  // ** récupère la Rangée modifiée
    Gsetcellcolor(X,Y,ClRed); // ** Colorie le fond de la cellule modifiée
  End;
End;
```

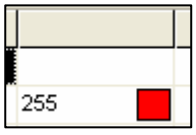
Autorise ou non la saisie dans un cellule

```
// *** Window1 ************************************
SObject Procedure Window1;
Var
 X,Y : Integer;
Begin
  GGetcurrent(X,Y); // ** récupère la position du curseur dans la grille
  If (X=1) and (Y=2) Then Gdevalidedit // ** Interdit Si curseur est en Col1 et
                     Else Gvalidedit;  // Rangée2 Autorise la saisie dans
                                       // toutes les autres cellules
End;
```

Accepte le drag & drop venant d'une zone de saisie

```
// *** Edit1 **************************************
SObject Procedure Edit1;
Begin
  Drag(Gettext); // ** Le drag & drop provient de la zone de saisie Edit1
End;
// *** Window1 ************************************
SObject Procedure Window1;
Var
 X,Y : IntegeTS : String;
Begin
  if Drop Then // ** Drag relâché sur la grille
  Begin
    Flyingcells(X,Y); // ** Cellule survolée par la souris
    Gsettext(X,Y,Getdrop);
  End;
End;
```

Appel d'une palette de couleur depuis une cellule



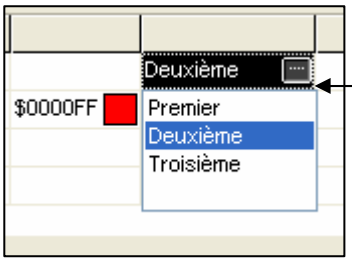

```
// *** Window1 ************************************
SObject Procedure Window1;
Begin
  if Start_visu then
  Begin
    GCommand('Color',2,2);        // ** La cellule est une cellule couleur
    GSetText(2,2,'$0000FF');      // ** Peint la cellule en Rouge ***
    GSetControl(2,2,'Palette');  // ** La cellule permet par double-clic
                                 //** d'appeler la  palette de couleur
  end;
End;
```

Afficher une listbox dans une cellule

Il est possible d'afficher des contrôles Windows (ComboBox / ListBox /Cases à cocher) dans les cellules.
Ces contrôles sont avant tout créés dans le synoptique à l'aide de l'éditeur graphique avec leurs différents types possible (DropDown/Simple/personnalisé/trié ou non etc..).
Un programme gérera le contenu et le choix de l'opérateur dans la liste avant de l'affecter à la cellule.
Un nom est proposé par l'éditeur graphique (Combo1,Switch2 Etc..), ce nom nous servira pour affecter ce contrôle à une cellule particulière.
Le contrôle doit être caché au lancement de l'application à l'aide de la commande **HideControl**. La grille le fera apparaître dans la cellule concernée. Le contrôle doit prendre en charge le click de la souris, la récupération de la chaîne de caractère et l'affectation de celle-ci à la cellule appelante.

// Création du programme dans la listbox qui doit apparaître dans la cellule de la grille.

```
// *** ListBox1 ************************************
SObject Procedure ListBox1;
Begin
 If Start_Visu Then     //** au lancement de l'appli
 Begin
    Addstring('Premier');   // Ajout du texte 'Premier' dans la listbox
    Addstring('Deuxième');  // Ajout du texte 'Deuxième' dans la listbox
    Addstring('Troisième'); // Ajout du texte 'Troisième' dans la listbox
    HideControl;            // Cacher la listbox
  End;
// Lorsqu'un clic souris est effectué dans la listbox le texte est écrit dans
// la cellule courante de la grille
  If  TopLBdown then
    GSetTextCurrent(GetSelString);  // Récupère le texte et l'envoi dans la
End;                                //  cellule courante


// *** Window1 ************************************
SObject Procedure Window1;
Begin
  if Start_visu then
  Begin
    GCommand('Color',2,2);    // *** La cellule est une cellule couleur
    GSetText(2,2,'$0000FF'); // *** Peint la cellule en Rouge ***
    GSetControl(2,2,'Palette');  // *** La cellule permet par double-clic
                                 // ***   d'appeler la palette de couleur
    GCommand('CONTROL',3,1);     // ** Colonne 3 et Rangée 1 contient un contrôle
    GSetControl(3,1,'Listbox1'); // ** le contrôle est la listbox1
  End;
End;
```

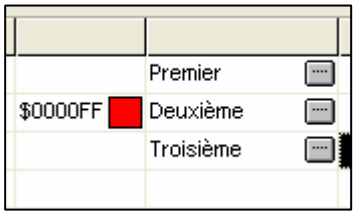


Lorsqu'une cellule est affectée par un contrôle, un petit bouton à droite de celle-ci apparaît permettant ainsi d'afficher un choix.

En double cliquant sur la cellule (3,1) la listbox apparaît sous la cellule affichant les lignes préprogrammées. Le programme dans la listbox1 étant actif, lorsque l'opérateur clique celle-ci elle écrit le texte dans la cellule puis la grille cache automatiquement cette Listbox.

Afficher la même listbox dans plusieurs cellules

```
// ** La même listbox peut être utilisée autant de fois que souhaitée.
// *** Window1 ****************
SObject Procedure Window1;
Begin
  if Start_visu then
  Begin
    GCommand('Color',2,2);       // *** La cellule est une cellule couleur
    GSetText(2,2,'$0000FF');     // *** Peint la cellule en Rouge
    GSetControl(2,2,'Palette'); // La cellule permet par double-clic d'appeler la
                                 // palette de couleur
    GCommand('CONTROL',3,1);     // ** Colonne 3 et Rangée 1 contient un contrôle
    GSetControl(3,1,'Listbox1'); // ** le contrôle est la listbox1
    GCommand('CONTROL',3,2);     // ** Colonne 3 et Rangée 2 contient un contrôle
    GSetControl(3,2,'Listbox1'); // ** le contrôle est la listbox1
    GCommand('CONTROL',3,3);     // ** Colonne 3 et Rangée 3 contient un contrôle
    GSetControl(3,3,'Listbox1'); // ** le contrôle est la listbox1
  End;
End;
```



Centrer les textes dans les colonnes

Utilisez la procédure **GSetOptions** permet de fixer les options d'affichage de la grille.

Différentes options sont possibles soit par la boite de propriétés (voir ci-avant) ou par passage de paramètres dans la procédure GSetoptions avec les combinaisons de constantes ci-dessous qui sont déclarées dans Windows.pas.

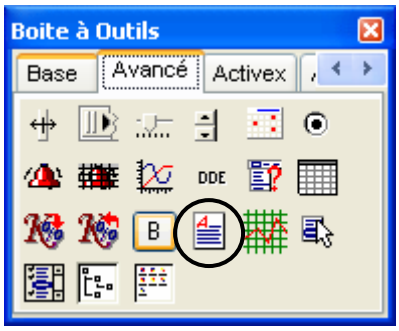
```
Const
 AlignCenter  = 16384;
 AlignRight   = 32768;

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;
```

Exemple : Derrière la grille de données, modifications des attributs. Les textes seront centrés dans les colonnes.

```
Uses Windows; //*** Pour utilisation des constantes prédéfinies
{*** Window1 ************************************}
SObject Procedure Window1;
Begin
  If Start_Visu Then  //*** Une fois au lancement de l'application
   Gsetoptions(GridOptions1 + AlignCenter );
End;
```

☞ **Note** Pour informations complémentaires sur GSetoptions, voir instructions système grilles de données.

Richedit – éditeur de texte formaté de Windows

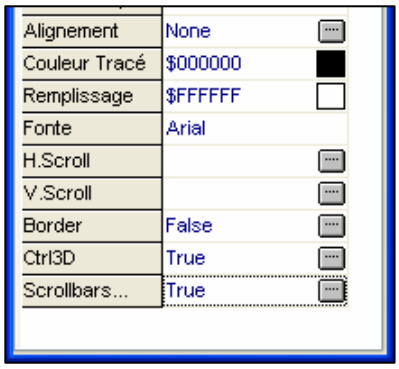


Le contrôle éditeur de texte formaté est un composant mémo qui gère le texte mis en forme, c'est-à-dire que vous pouvez modifier individuellement le formatage des caractères, des mots ou des paragraphes.
Les contrôles éditeur de texte formaté proposent également des fonctions d'impression et de recherche de texte.
Par défaut, l'éditeur de texte formaté gère :
Les propriétés de fonte, comme la police, la taille, la couleur, les formats gras ou italique.
Les propriétés de format comme l'alignement, les tabulations, l'indentation ou la numérotation.
Le *glisser-déplacer* automatique du texte sélectionné.
L'affichage de texte formaté ou de texte brut.

Les procédures et fonctions pour manipuler un richedit sont contenue dans la librairie **RICHINT.VPU** dont le source est fourni avec Visual I/O. Ce reporter au chapitre librairie RICHINT

La boite de propriété d'un Richedit est restreinte et ressemble à celle ► d'une fenêtre interne.

La propriété **Scrollbars** détermine s'il faut ajouter des barres de défilement au contrôle quand le texte est trop large ou trop long pour un affichage complet.
Les propriétés Couleur tracé, Fonte, H.Scroll, V.Scroll sont inactives.



La librairie Richint

Cette librairie est décrite en détail dans le *chapitre librairie Richint*. Elle permet à l'aide de procédures et de fonctions de piloter un contrôle Richedit. Extrait de la liste de ses procédures :

```
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** : Se reporter au chapitre pour les détails de chaque procédure.

Programmation sur les richedit

Charger un fichier dans un contrôle richedit
Sauve le contenu d'un contrôle richedit dans un fichier

```
Uses Richint;
// *** Window1 ************************************
SObject Procedure Window1;
Begin
  // ** Charge le fichier dans le Richedit
  If Openf  Then RichLoadFromFile ('C:\Description.Rtf');
  // ** Sauve le RichEdit dans le fichier
  If Savef  Then RichSaveToFile   ('C:\Descriptionx.Rtf');
  If ClearF Then RichClear(Gethwnd); // ** Efface le richedit
  OpenF:=False;
  Savef:=False;
  ClearF:=False;
End;
```

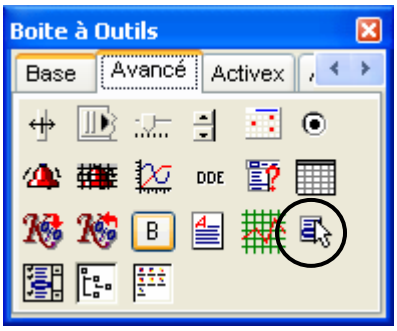
☞ **Note** : Si la procédure **RichLoadFromFile** et **RichSaveToFile** est utilisée derrière le composant richedit seulement un seul paramètre peut être passé car la procédure travaille sur le richedit courant. Dans le cas ou le fichier doit être chargé dans un autre Richedit, il faut passer un paramètre supplémentaire qui est le handle du richedit concerné.

Les paramètres optionnels sur les procédures sont disponibles à partir de la version 7.00 de Visual I/O.

Charger un fichier dans un autre richedit

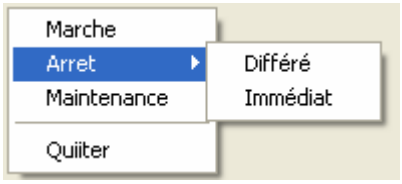
```
Var
  HDR : Integer ;
// ** récupère le handle du richedit au lancement de l'appli
// *** Window2 ************************************
SObject Procedure Window2;
Begin
  If Start_visu then HDR:=GetHwnd;
End;

// *** Button1 ************************************
SObject Procedure Button1;
Begin
  // ** Charge le fichier dans un Richedit de hndle HDR
  // ** HDR handle cible
  If Openf Then RichLoadFromFile ('C:\Description.Rtf',HDR);
End;
```

PopupMenu - Menu surgissant



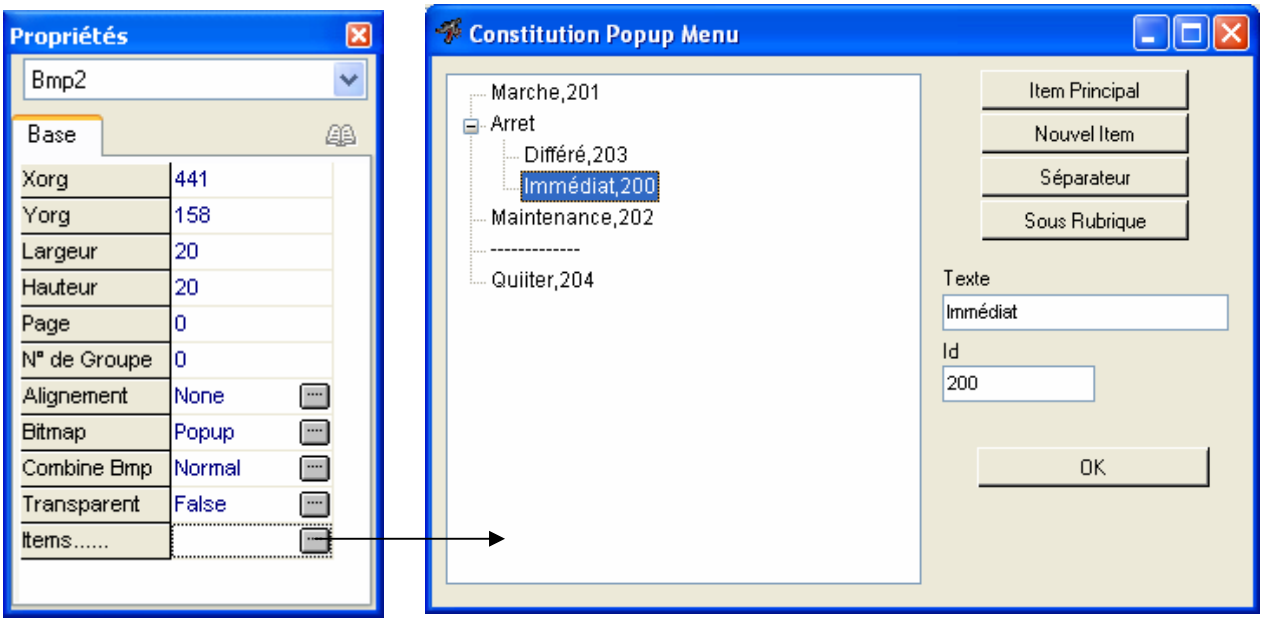
Le contrôle PopupMenu permet de définir le menu surgissant qui apparaît quand l'utilisateur clique avec le bouton droit de la souris.

Exemple de menu surgissant incluant une sous rubrique Arrêt



Concepteur de menu surgissant

Placez le composant sur le synoptique, puis frappez <F6> pour appeler la boite de propriétés. Seul le paramètre Items est utile. Double-cliquez sur celui-ci pour appeler le concepteur de menu flottant.



Le principe de création des items d'un menu surgissant est similaire au menu principal.
Des identificateurs sont associés aux textes du menu flottant permettant ainsi un test par programme même si ces textes varient en fonction de la langue utilisée. Les identificateurs s'incrémente automatiquement mais démarre à 200 pour le premier item. Il est possible de modifier cet identificateur dans le champ de saisie Id.

Programmation sur les menus flottants

Valider ou non un item de menu

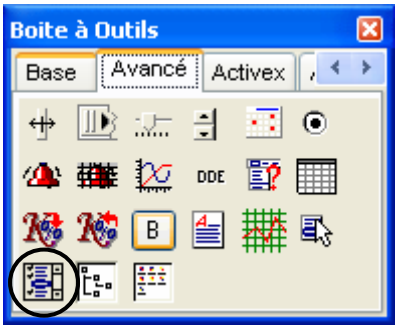
```
// *** Bmp2 ************************************
SObject Procedure Bmp2;
Begin
 If Go Then ValidMenuItem(201,True)
       Else ValidMenuItem(201,False);
End;
```

Ajouter un bitmap devant un item de menu


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

☞ **Note** : Le bitmap doit être de 16 couleurs avec la couleur transparente qui est donnée par le pixel en bas à gauche du bitmap. Ce bitmap peut être inclus dans l'exécutable ou présent dans le répertoire du projet.

CheckListBox - liste de cases à cocher



Affiche une liste défilante contenant une case à cocher à côté de chaque élément.

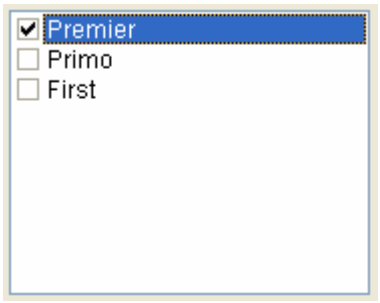
Les utilisateurs peuvent cocher ou décocher les éléments de la liste.

Différences avec les listbox classiques

Le contrôle CheckListBox est une version simplifiée de la listbox.
Ce contrôle permet les opérations courante d'une listbox : Ajout, insertion, effacement de lignes .. plus la possibilité de cocher/décocher les case et de tester leurs états.

Programmation sur les checklistbox

Cocher ou décocher un item dans la liste




```
// *** ListBox1
SObject Procedure ListBox1;
Begin
  If Start_visu Then
  Begin
    Addstring('Premier'); // Ajoute des lignes à la
    Addstring('Primo');   // Listbox
    Addstring('First');
    CheckListSetState(0,True); // ** Coche la première
    CheckListSetState(1,False); // Ligne
    CheckListSetState(1,False); // ** Décoche la deuxème ligne
  end;
End;
```

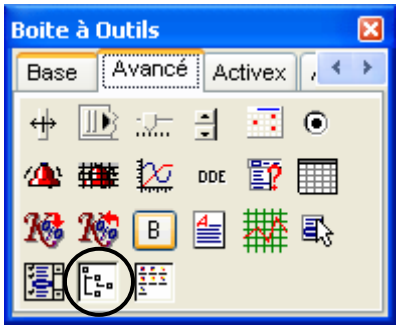
Test la case à cocher d'une ligne dans la listbox

```
// *** ListBox1 ************************************
SObject Procedure ListBox1;
Begin
   Go:=CheckListGetState(0); // ** Go=True si la case est cochée
End;
```

Test si au moins une case est cochée dans la listbox

```
// *** ListBox1 ************************************
SObject Procedure ListBox1;
Begin
  Go:=False;
  For I:=0 to GetCount-1 do
  Begin
    If CheckListGetState(I) Then Go:=True; // ** test si la case est cochée
  End;
End;
```

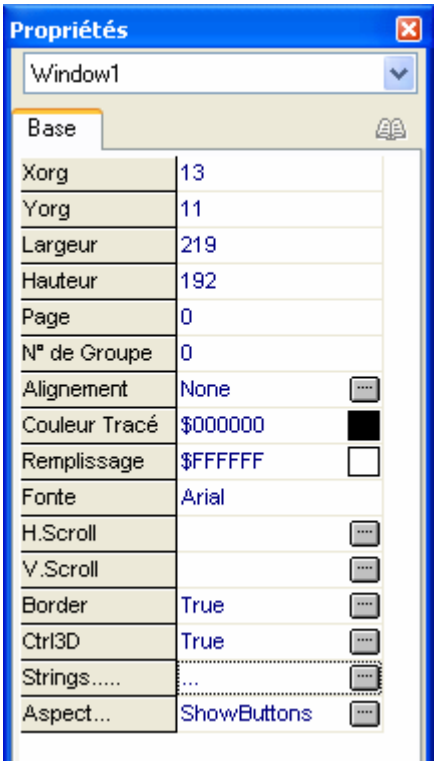
TreeView – liste triée

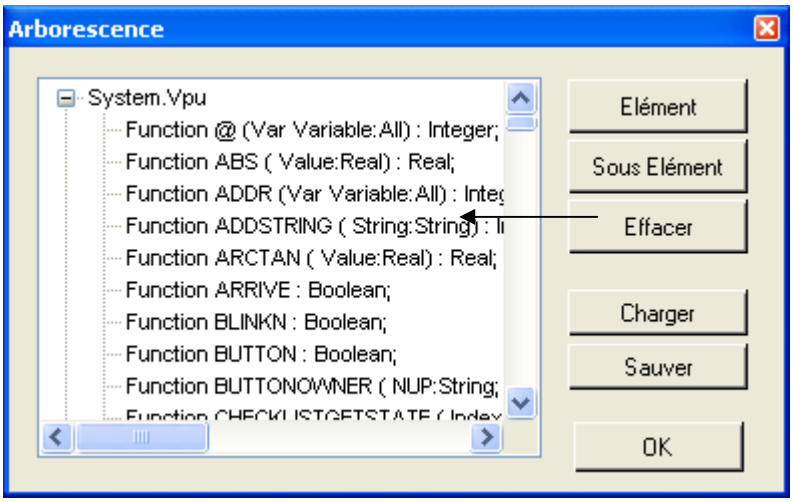


Permet de contrôler et d'afficher un ensemble d'objets sous forme d'arborescence indentée, en se basant sur leur relation hiérarchique logique.
Le contrôle comprend des boutons qui permettent de réduire ou de développer l'arborescence.
Ce composant permet d'afficher la relation entre un ensemble de conteneurs ou leurs éléments hiérarchiques.

Description

Utilisez TreeView pour ajouter un développement et visualiser une arborescence dans un synoptique. Chaque nœud d'un TreeView consiste en un libellé avec des images bitmap facultatives. Chaque nœud peut contenir une liste de sous nœuds associés. En cliquant sur un nœud, l'utilisateur peut développer ou réduire la liste associée de sous nœuds. Le volet gauche de l'explorateur de l'éditeur de Visual I/O propose un exemple de ce contrôle.





Le TreeView ne contient pas d'expert. Il faut coder pour l'utiliser.

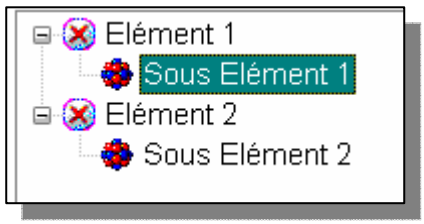
Double-cliquez sur string, un concepteur d'arborescence permet de créer des noeuds incluant ou non des sous éléments.
Ce concepteur permet aussi de charger et sauver en design le contenu du treeview dans un fichier Texte.

La propriété Aspect permet l'affichage ou non des boutons pour développer les arborescences
Pour modifier le texte d'un nœud ou d'un sous élément, sélectionnez le puis cliquez une seconde fois dessus. Celui-ci se transforme en champ de saisie afin de le modifier.

Programmation sur les treeview

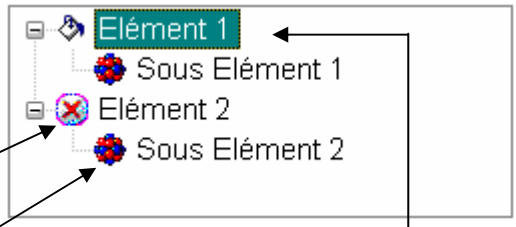
Ajouter des images dans un treeView

***Attention* : Les images doivent êtres de même taille !!**




```
// *** ListBox1
SObject Procedure Window1;
Begin
  If Start_visu Then
  Begin
    Xaddimage('Closer16'); // ** Bitmap sur tous les noeuds de niveau 0
    Xaddimage('ATOMES');   // ** Bitmap sur sous tous les éléments de niveau 1
  End;
End;
```

Afficher une image différente à un élément de treeView




```
// *** ListBox1 *****************
SObject Procedure Window1;
Begin
  If Start_visu Then
  Begin
    Xaddimage('Closer16'); // ** 1ère Image Niveau 0
    Xaddimage('ATOMES');   // ** 2ème Image  Niveau 1
    Xaddimage('MenuPalette'); // ** 3ème Image Niveau 2 inexistant ici
    Xsetimage(1,2);           // ** Forçage Premier élément de l'image 2
  End;
End;
```

☞ **Note** : Les éléments sont numérotés à partir 1 (le plus haut) vers l'élément le plus bas.

Connaître l'élément sélectionné et le nombre d'éléments dans le treeview

```
// *** ListBox1 **************
SObject Procedure Window1;
Begin
 Selection:=TreeGetSelectedItem; // ** retourne le Numéro de ligne sélectionnée
 Count:= TreeGetCount;  // ** retourne le nombre de lignes (items) dans le treeview
End;
```

Modifier le texte d'une ligne

```
// *** ListBox1 *****************
SObject Procedure Window1;
Begin
  If TreeGetItem(1)<>'ARSOFT' Then //** Lit la première ligne et compare le texte
     TreeSetItem(1,'ARSOFT'); // ** Affecte un nouveau texte sur la ligne N°1
End;
```

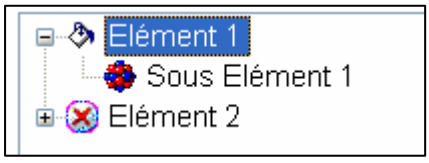
Rechercher un texte dans un TreeView et forcer la sélection

```
// *** ListBox1 ************************************
SObject Procedure Window1;
Begin
  For i:=0 to TreeGetCount-1 Do // ** Scan toutes les lignes
  Begin
    If TreeGetItem(I)='ARSOFT' Then // ** Compare la ligne au texte ARSOFT
       TreeSetSelectedItem(I);     // Si trouvé force la sélection sur la ligne
  End;
End;
```

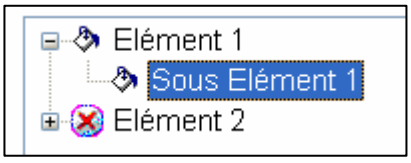
☞ **Note** : Le fait de forcer une sélection affiche la ligne (ouverture de l'arborescence si nécessaire)

Fixe une image à un nœud seulement quand il est sélectionné

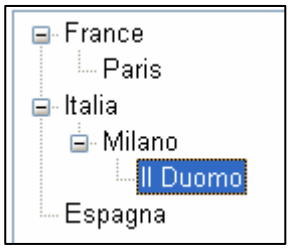
```
SObject Procedure Window1;
Begin
  If Start_visu Then
  Begin
    Xaddimage('Closer16'); // ** 1ère Image Niveau 0
    Xaddimage('ATOMES');   // ** 2ème Image  Niveau 1
    Xaddimage('MenuPalette');  // ** 3ème Image Niveau 2 inexistant ici
    Xsetimage(1,2);             // ** Forçage Premier élément de l'image 2
    TreeSetSelectedImage(2,2); // L'image N° 2 est affiché quand nœud 2 est sélectionné
  End;
End;
```



Changement d'image quand la sélection est sur la ligne N°2



Effacer puis ajouter des lignes dans un treeview




```
SObject Procedure Window1;
Begin
  if Start_visu then
  Begin
    TreeLoadFromFile('XXXX');    // Efface le contenu du treeview quand le fichier
    TreeAdd(0,'France');          // est inexistant
    Treeaddchild(1,'Paris');     // Ajoute à la ligne 1 une sous rubrique
    TreeAdd(0,'Italia');         // Ajoute à la fin (0) un nouvelle rubrique
    Treeaddchild(3,'Milano');    // Ajoute à la ligne 3 une sous rubrique
    Treeaddchild(4,'Il Duomo'); // Ajoute à la ligne 4 une sous rubrique
    TreeAdd(0,'Espagna');        // Ajoute à la fin une rubrique
  End;
End;
```

Trier un treeview

La procédure XAlphaSort permet de trier la totalité du treeview.

```
    ...
    Treeaddchild(4,'Il Duomo');
    TreeAdd(0,'Espagna');
    XAlphaSort ;
End;
```

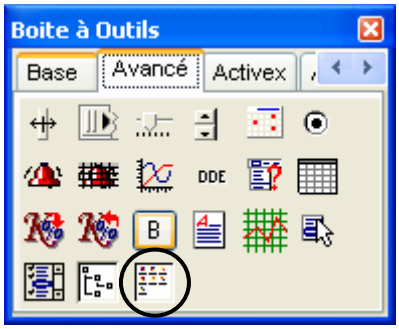
Ajouter des lignes dans un treeview utilisant un fichier

```
// *** ListBox1 ******
SObject Procedure Window1;
Var
  Fic : Integer;
Begin
  if Start_visu then
  Begin
    FileDelete('Temp');
    Filetextappend(Fic,'TEMP'); // ** Créer un fichier vide en mode ajout
    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'); //  Charge le Fichier constitué
    XAlphaSort;
  End;
  If Go Then TreeSaveToFile('MODELE.TXT'); // Sauve le treeview dans un fichier
End;
```

Refermer complètement un treeview

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

Listview - Liste d'éléments

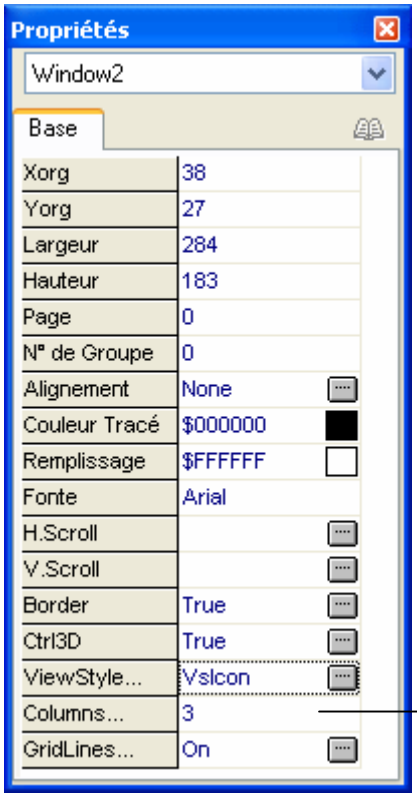


Utilisez ListView pour gérer et afficher une liste d'éléments dans un synoptique.
Les éléments peuvent être affichés en colonnes avec des en-têtes et des sous éléments, verticalement ou horizontalement avec de petites ou de grandes icônes.

Description

Les éléments du contrôle ListView permet peuvent être affichés comme un groupe d'icônes mobiles ou comme des colonnes de texte. Les valeurs possibles sont :

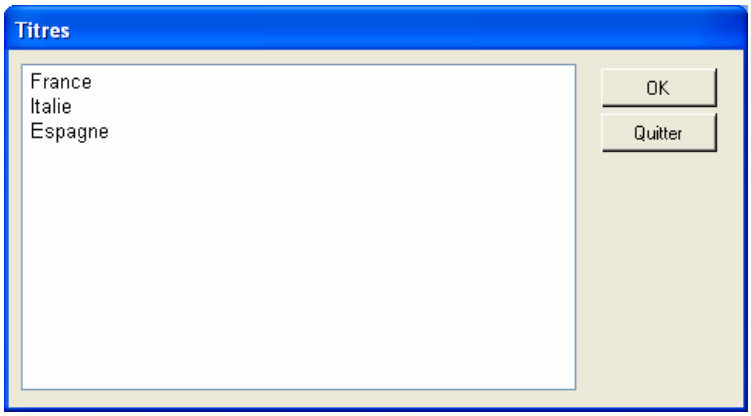
vsIcon Chaque élément apparaît comme une icône de taille normale avec un libellé en dessous. L'utilisateur peut faire glisser les éléments n'importe où dans la fenêtre de la vue liste.
vsList Chaque élément apparaît comme une petite icône avec un libellé à sa droite. Les éléments sont organisés en colonnes et ne peuvent être déplacés par l'utilisateur.
vsReport Chaque élément apparaît dans une ligne distincte avec ses informations organisées en colonnes. La représentation est similaire à une grille de données.



Vsicon

VsList

VsReport

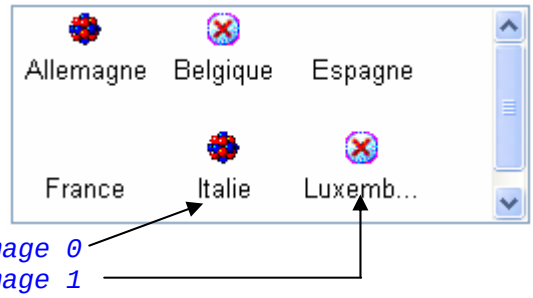


Colums permet de pré définir les textes qui seront affichés dans les 3 modes possibles.
Gridlines permet d'afficher les lignes du rapport dans le mode VsReport.
Les icônes sont enregistrées et fixées par programme uniquement.

Programmation sur les ListView

Fixer des images devant les textes (mode Vsicon et VsList)

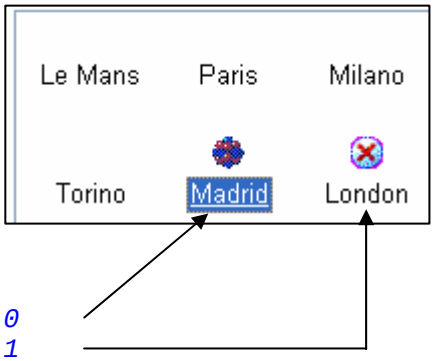
```
// *** Window2 ***************
SObject Procedure Window2;
Begin
 if Start_visu Then
 Begin
   Xaddimage('Atomes');   // Image 0
   Xaddimage('Closer16'); // Image 1
   XSetImage(4,0); // ** Affecte au texte 4 l'image 0
   XSetImage(5,1); // ** Affecte au texte 5 l'image 1
 End;
End;
```



☞ **Note** : A chaque enregistrement d'image, celles-ci sont affectées automatiquement en numéro d'ordre des textes. Ici L'image 0 est affectée directement au texte 0, l'image 1 au texte 1.
L'instruction **XSetImage** modifie cette affectation en désignant un autre numéro d'image au texte spécifié.

Effacer et ajouter des lignes dans une listview

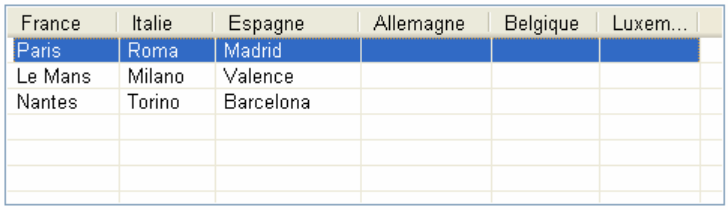
```
// *** Window2 ********
SObject Procedure Window2;
Begin
 if Start_visu Then
 Begin
   Xaddimage('Atomes'); // ** Image 0
   Xaddimage('Closer16'); // ** Image 1
   XSetImage(4,0); // Affecte au texte 4 l'image 0
   XSetImage(5,1); // Affecte au texte 5 l'image 1
     ClearList;    // Efface la liste entière
     Addstring('Le Mans'); // Ajoute les textes
     Addstring('Paris');
     Addstring('Milano');
     Addstring('Torino');
     Addstring('Madrid');
     Addstring('London');
     XSetImage(4,0); // Ré Affecte au texte 4 l'image 0
     XSetImage(5,1); // Ré Affecte au texte 5 l'image 1
 End;
End;
```



Les images précédentes restent enregistrées mais non sont plus affectées automatiquement. Il faut alors utiliser la fonction **XSetimage** pour refaire toutes les affectations nécessaires.

Ajouter des lignes dans une listview type report

```
SObject Procedure Window2;
Begin
 if Start_visu Then
 Begin  // ** ajout dans les colonnes
   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;
```



Chaque colonne peut être triée en cliquant sur son titre.
La première colonne (à gauche) porte le N° 0.

☞ **Note** : la procédure Addstring fonctionne mais ne remplie que la première colonne.

Rechercher un élément dans une listview type report

```
For X:=0 To 2 Do
 For Y:=0 To GetCount Do // ** Ligne 0 = Titre
   Begin
     TS:=LvGetitem(X,Y);
     If TS='Paris' Then
       SetSelIndex(Y); // ** Fixe la sélection sur la ligne contenant 'Paris'
   End;
```

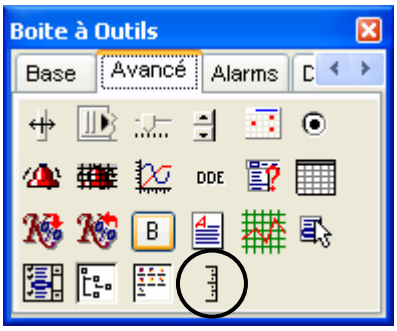
Fonction et procédure communes à la listbox et à la listview

```
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;
```

Les procedures et fonctions suivantes concernant les grilles fonctionnent aussi sur un report

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

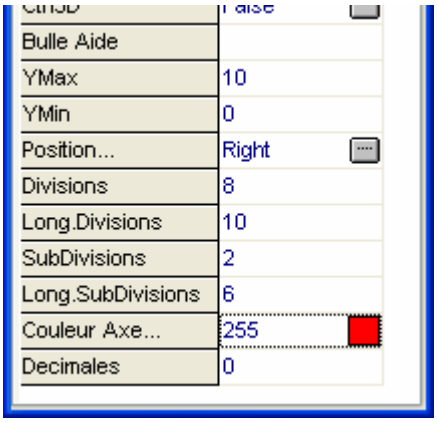
Axe additionnel



Utilisez le composant axe pour afficher un axe supplémentaire à coté d'une courbe, bargraph slider etc.

Description

La boite de propriété permet de régler les différents attributs ainsi que ceux spécifiques à l'axe.



YMax : Valeur de la graduation la plus haute (Maxi).
YMin : Valeur de la graduation la plus basse.
Position : Right ou Left. Graduations dessinées à gauche (left) ou à droite (right).
Divisions : Nombre total de divisions principales.
SubDivisions : Nombre total de sub divisions.
Long.Divisions : Longueur exprimée en pixel du trait de division.
Long.SubDivisions : Longueur exprimée en pixel du trait de sub-division.
Couleur Axe : Couleur de tracé de l'axe, des divisions et des sub-divisions.
Decimales : Nombre de décimales à afficher.

Programmation sur les axes

Vous pouvez modifier dynamiquement les attributs de l'axe à l'aide de la librairie axisint.pas (livrée avec code source).

AxisYMax : Variable **Real** fixant la valeur maximum de l'axe.

AxisYMin : Variable **Real** fixant la valeur minimum de l'axe.

AxisRight : Variable **Boolean** fixant le trace à droite ou à gauche de l'axe.

AxisDiv : Variable **Integer** fixant le nombre de division de l'axe.

AxisSubDiv : Variable **Integer** fixant le nombre de sub division de l'axe.

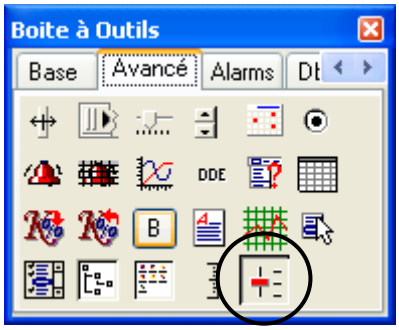
AxisColor : Variable **Integer** fixant la couleur de tracé de l'axe.

AxisDecimals : Variable **Integer** fixant le nombre de décimales tracées sur l'axe.

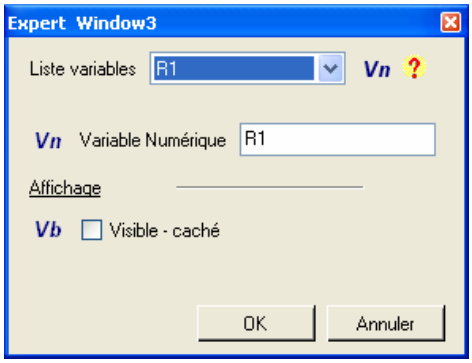
Changer la valeur maximum et minimum de l'axe

```
If Changer Then
Begin
  Go2('Window1');      //*** Va sur le composant Axe
  AxisYMax:=NewYMax;   //*** Affecte la nouvelle valeur Maxi de l'axe
  AxisYMin:=0;         //*** Affecte la nouvelle valeur Mini de l'axe
End;
```

Potentiomètre additionnel



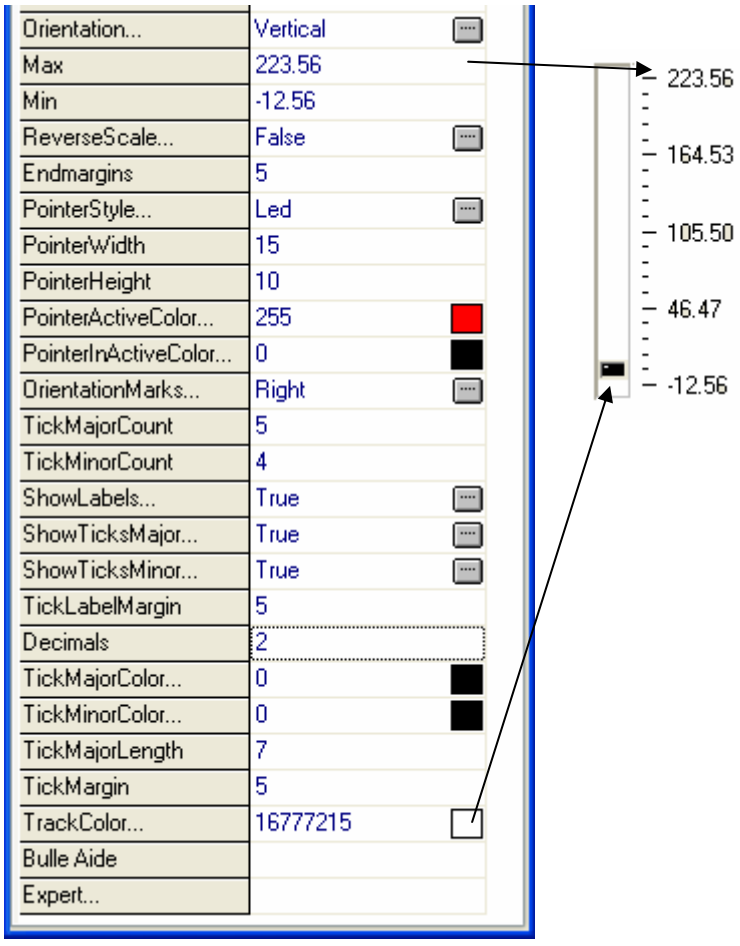
Utilisez ce composant pour fixer une valeur de type Integer ou réel à l'aide d'un potentiomètre. Ce composant est plus complet que le slider de windows. Permettant plusieurs aspects.



L'expert permet le paramétrage d'une variable integer ou réel. Cet expert est aussi celui d'une valeur numérique

Description

La boite de propriété permet de régler les différents attributs ainsi que ceux spécifiques au potentiomètre.

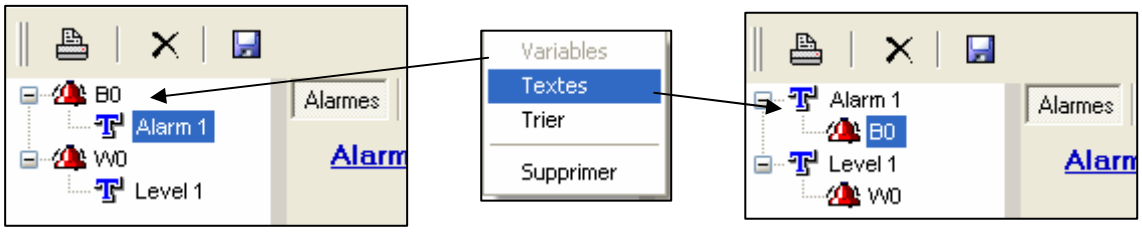


Orientation : verticale ou Horizontale.
ReverseScale : Mini en haut ou en bas.
EndMargins : Décalage en pixel des premières graduations en haut et en bas.
PointerWidth
Pointer Height : Largeur et hauteur curseur en pixels.
PointerActiveColor : Couleur du curseur quand l'opérateur pousse celui-ci.
PointerInActiveColor : Couleur du curseur quand l'opérateur ne pousse pas celui-ci.
OrientionMarks : Graduations à droite ou a gauche.
Showlabels, ShowTickMajor, ShowTicksMinor : Les graduations et sous graduations sont visibles ou non.
TickLabelMargin : décalage en pixels des textes par rapport aux graduations.
Decimals : Nombre de décimales affichées sur les graduations.
TickMajorColor, TickMinorColor : Couleurs des graduations et des sub divisions.
TickMajorLength : Longueur en pixels des graduations principales.
TickMargin : décalage en pixels des graduations par rapport à la glissière du curseur.
TrackColor : Couleur de la glissière.

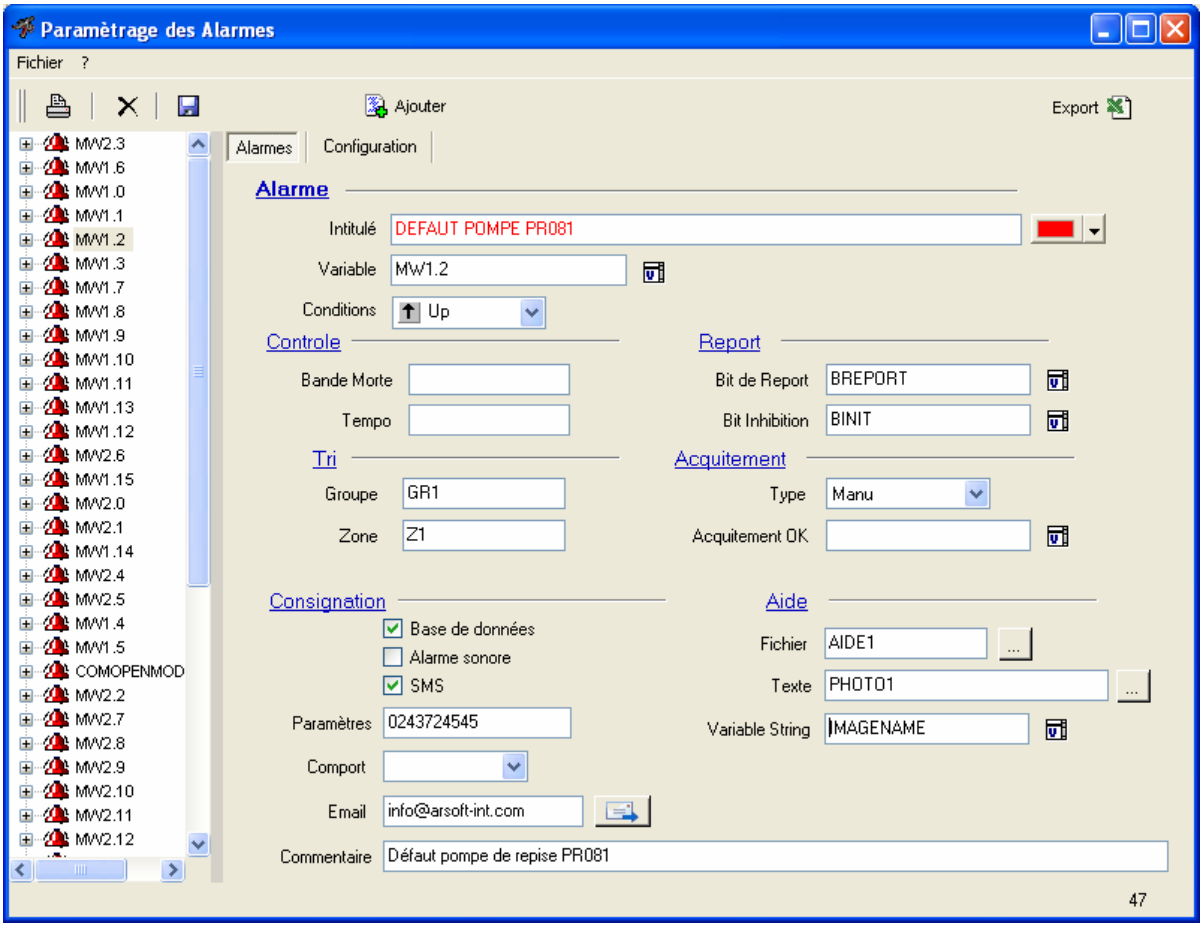
Gestion des Alarmes

Dans le menu principal de l'éditeur Visual I/O

Un clic droit sur le navigateur permet de passer d'une liste de variables à une liste d'intitulés.
Le tri alphabétique est disponible dans les 2 cas.



Fenêtre principale



Section Alarme

intitulé : Texte de l'alarme consignée sur écran, Base de données Imprimante email ou liaison série ..
Variable : Variable Bit ou mot à tester déclenchant l'alarme.
Conditions : Conditions déclenchant l'alarme.
Up = Sur Front montant de la variable.
Down = Sur Front descendant de la variable.
>,<,= à une constante ou une variable.

Section contrôle

Bande Morte : Permet de créer une zone d'hystérésis autour de **la valeur numérique** à tester.
Ceci permet d'éviter les déclenchements successifs dus à des fluctuations de la valeur testée.

La bande morte n'est valide que pour une alarme numérique. Si la valeur à tester est 100 et que la bande morte est égale à 4. La variable globale numérique pourra fluctuer de 98 à 102 sans redéclenchement d'alarme.

Tempo exprimé en ms elle permet de temporiser l'alarme.
Si au bout de cette tempo l'alarme est toujours présente alors elle est réellement prise en compte par le système.
Ceci évite les déclenchements intempestifs ou alarmes fugitives.

Section Report

Bit de report : Bit global passant à un 1 quand l'alarme ou une des alarmes est présente.
Le même bit peut être affecté à plusieurs alarmes.

Bit Inhibition : Bit global (facultatif) permettant d'interdire le déclenchement de l'alarme.
Si ce champ est vide l'alarme est toujours en service.

Section Tri

Groupe, Zone : Texte simple permettant de structurer les alarmes par appartenance et permettant un tri plus facile de celles-ci.

Section Acquittement

L'acquittement peut être Automatique ou Manuel.

Automatique : Dès que l'alarme disparaît, elle ne sera plus visible dans les ListBoxes des synoptiques.
Dans ce cas la date et l'heure d'acquittement correspond à celles de la disparition.

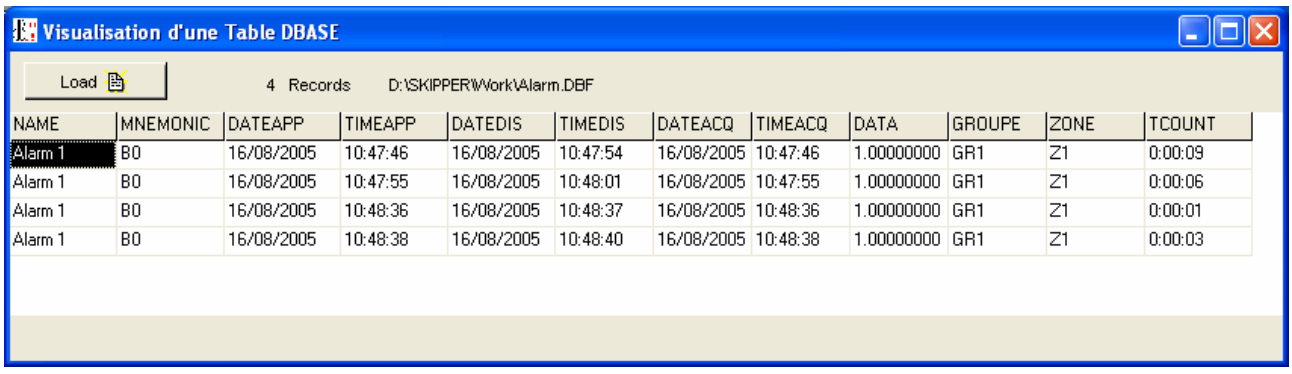
Manuel : Quand l'alarme disparaît le texte reste visible dans les ListBoxes des synoptiques avec le petit bitmap symbolisant une 'Cloche' coché.
L'opérateur devra cliquer dessus pour acquitter cette alarme. La date et l'heure d'acquittement seront alors enregistrées.

Section Consignation

Base de données : Les alarmes sont consignées dans le fichier Alarm.dbf (Format DBase 3+). Ce fichier est lisible par les logiciels de bureautique type excel.

Enregistrements d'une table DBase d'alarmes :

Sont stockées : Date et Heure d'apparition, date et heure de disparition, date et heure d'acquittement et le temps de présence (TCount). Les autres informations sont la valeur de déclenchement de l'alarme (Data) le groupe et la zone si ceux-ci ont été paramétrés.



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** : *L'utilitaire VisuDBF (ci-dessus et créé avec Visual I/O) permet de visualiser et de modifier une table Dbase.*

Alarme sonore Permet d'activer le Buzzer du PC en cas d'alarme.

Comport Permet de désigner le n° de Com permettant d'envoyer le texte de l'alarme sur la liaison série.
La configuration de la liaison série (Vitesse, Data bits etc..) est faite dans l'onglet Configuration.

Email Permet de spécifier une adresse Email sur laquelle sera envoyée le texte de l'alarme.
Vous devez pour cela **spécifier un Serveur SMTP**. Ce serveur est celui spécifié dans Outlook par Exemple (smtp.fournisseur d'accès).
Dans le cas de Free : le serveur SMTP sera smtp.free.fr

Commentaire Texte interne d écrivant l'alarme.
SMS Permet d'appeler in programme externe mis en place par le programmeur (Envoie deSMS par exemple). Ce programme externe ne sera appelé si la case est cochée.
Paramètres Dans ce champ de saisie l'on peut placer une chaine de caractères quelquonce qui sera passée au programme externe.

Section Aide

Fichier Il est possible d'associer un fichier d'aide (format Texte) à chaque alarme que vous pourrez afficher en 'exploitation' mais ce fichier reste facultatif.
Frappez dans la cellule 'Fichier Aide :', le nom du fichier que vous voulez associer à l'alarme.

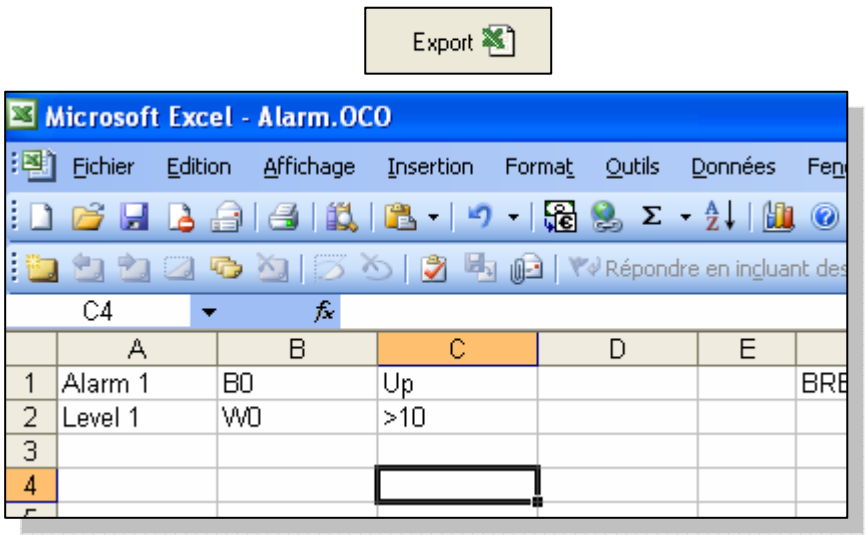
Cliquez sur le bouton à droite de ce champ de saisie pour appeler l'éditeur de texte de Windows (NotePad.exe).
Frappez le texte d'aide puis quittez NotePad.exe.
Le fichier spécifié se trouve alors dans la directorie du projet courant. Si vous utilisez Install Builder, ajoutez manuellement au projet ces fichiers texte (*.txt).

Texte Il est possible à chaque alarme d'associer une chaîne de caractères recevant un message prédéfini. Ce texte sera alors chargé à chaque sélection d'alarme dans une ListBox ou grille de visualisation d'alarmes. Ce texte pourra par exemple être le nom d'un bitmap, permettant alors de visualiser un photo localisant par exemple le défaut.

Dans l'exemple ci-dessus le texte 'Photo1' sera téléchargé dans la variable String IMAGENAME lorsque l'opérateur sélectionnera l'alarme dans la grille de visualisation des alarmes.

Export sous Excel

Cliquez sur le bouton Export. Celui-ci ouvre automatiquement Excel et affiche sous forme de lignes et colonnes toutes les alarmes. Le fichier de configuration des alarmes est ALARM.OCO dans le répertoire du projet courant.



Mise en place d'un appel à une procédure spécialisée type envoi de SMS.

1/ Pour chaque alarme souhaitée cochez la case SMS
2/ indiquez éventuellement un paramètre optionnel à récupérer dans la procédure externe spécialisée.

Mettre en place la procédure spécialisée.

La procedure spécialisée est écrite dans une librairie VPU ou dans un synoptique, vous disposez alors des sources pour toute modification ou évolution nécessaire.
Chaque alarme appellera donc cette procédure si la case SMS est cochée dans le paramètreur.

Pour mettre en place la procédure spécialisée il faut affecter la variable système **CallBackAlarmEx** qui est un pointeur sur une procedure qui recoit les paramètres.
La procedure avec ses paramètres est la suivante :

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

Date	: Date d'apparition de l'alarme
Time	: Heure d'apparition de l'alarme.
Intitule	: Texte de l'alarme.
Group et Zone	: Groupe et Zone paramétré.
Treatment	: Apparition = 1; Disparition = 2; Acquittement = 3.
Value	: Valeur qui à déclenchée l'alarme.
AlarmType	: Bit = 66 ('B') ou Valeur Numérique 78 ('N');

Exemple de mise en place dans un synoptique

```
//**** Procedure appellée à chaque apparition, disparition & acquittement d'alarme *****
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); //** Mise en place du Callback
     CallBackAlarmEx:=PT^;
   End;
End;
```

Envoi d'Emails Multiples

Vous devez écrire un petit programme pour intercepter la procedure de CallBack. Vous pouvez écrire ce code dans le synoptique principal de votre application

```
//**** Procedure appellée à chaque apparition, disparition & acquittement d'alarme *****
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

  //** Envoie de 3 emails vers différentes adresses
  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); //** Mise en place du Callback
     CallBackAlarmEx:=PT^;
   End;
End;
```

Envoi vers GSM

La librairie GSM.Pas et GSM.VPU permettent l'envoi de SMS message sur GSM.

Interface de la librairie :

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

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

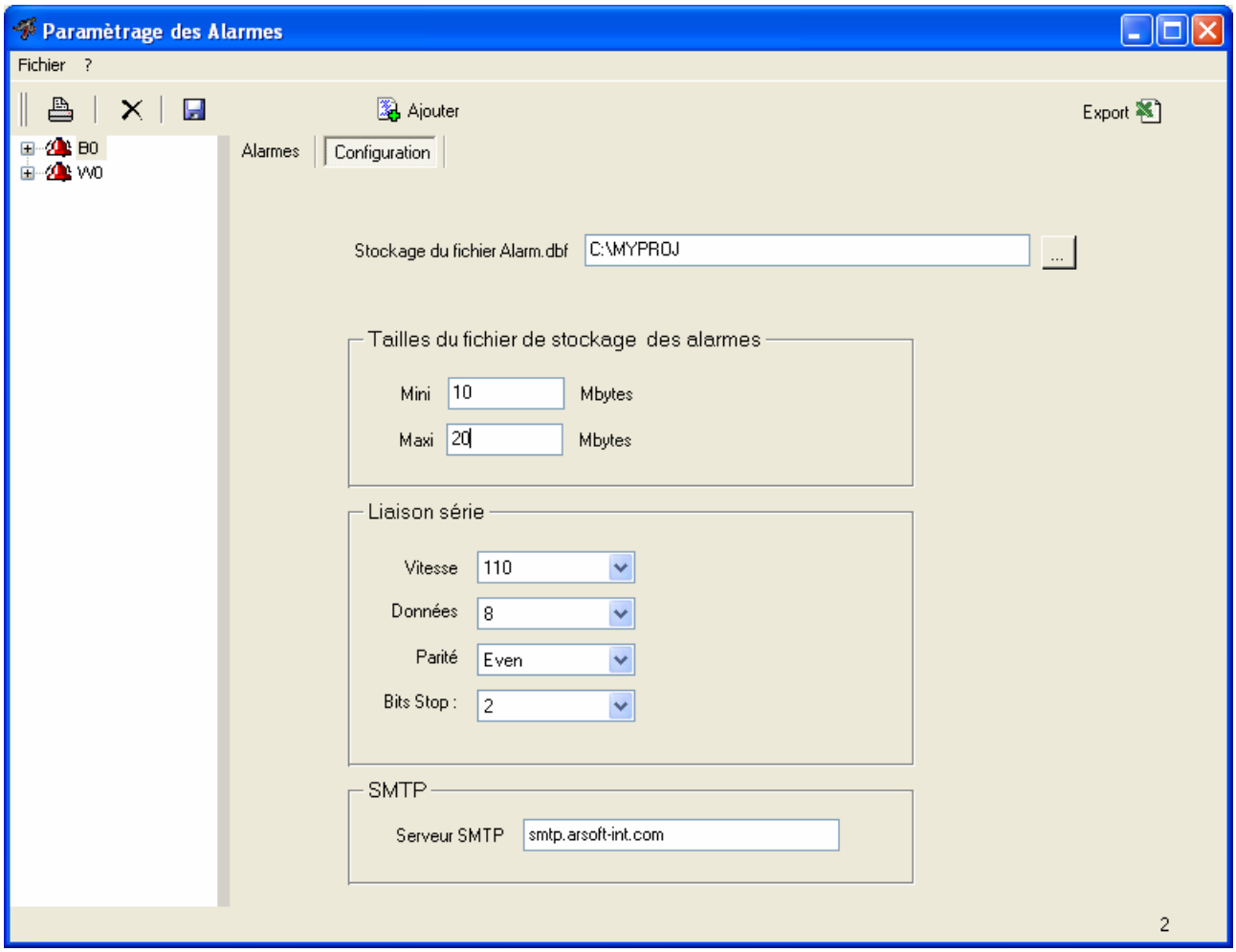
Cette function permet d'envoyer un texte sur un numéro de telephone (Phonenumber).
Cette fonction renvoi le nombre de SMS en attente dans le buffer d'émission de la librairie GSM.

La librairie inclue en interne une pile d'envoi de SMS permettant de mémoriser toutes les demandes en cas de transmission en cours ou d'occupation. Si un SMS ne passe pas le système le laisse en mémoire (buffer) et essaie d'envoyer le suivant dans la liste.

```
Uses GSM, Serial;

//* Procedure appellée à chaque apparition, disparition & acquittement d'alarme
Procedure MyCallBackProc (Date, Time, Intitule, Group, Zone: String;
      Treatment   : Integer;
      Value       : String;
      AlarmType   : Integer;
      Parameter   : String);
Var
  Sorte : String;
  Mess  : String;
Begin
   Case Treatment Of
      1 :  Sorte:='Apparition';
      2 :  Sorte:='Disparition';
      3 :  Sorte:='Acquittement';
  End;
  Mess:= Date+' '+Time+' '+Sorte+' '+Intitule;
  If AlarmType =78 then Mess:=Mess+Value; //*** Alarme Numérique
  Send_SMS (Mess, Parameter);
End;

{*** Button1 ************************************}
SObject Procedure Button1;
Var
 PT : Integer;
Begin
   If Start_visu Then
   Begin
     PT:=Addr(MyCallBackProc); //** Mise en place du Callback
     CallBackAlarmEx:=PT^;
     Opencom('1','9600','E','8','2');  //*** Ouverture du com port COM1
     GSM_COMPORT:=1;                   //*** Mémoire interne du com port
   End;
End;
```

☞ **Note** : Pour d'autre orientation des SMS il est aussi possible d'aiguiller en fonction du Groupe ou Zone de l'alarme.

Onglet Configuration



Stockage du fichier Alarm.dbf

Vous pouvez spécifier un répertoire pour le stockage des alarmes (en exploitation).
Si aucun fichier n'est spécifié, le répertoire de l'application finale est utilisée.

Section Tailles du fichier de stockage des alarmes

Permet de définir la taille mini et maximum du fichier de stockage des alarmes (Alarm.dbf) et de faire une purge automatique.

Maximum est exprimé en Méga bytes sera la taille maximum autorisée du fichier Alarm.dbf.. Quand celle-ci est supérieure ou égale à la valeur spécifiée, le système purge les alarmes les plus anciennes pour revenir à la taille **Minimum** demandée dans le champ Mini.

Section Liaison série

Vitesse, données, Parité et Bits stop sont les paramètres de la ou des liaisons série utilisées dans la section consignation du premier onglet. Toutes les liaisons ou la liaison série (en générale unique) sont paramètrées avec ces données.

Section SMTP

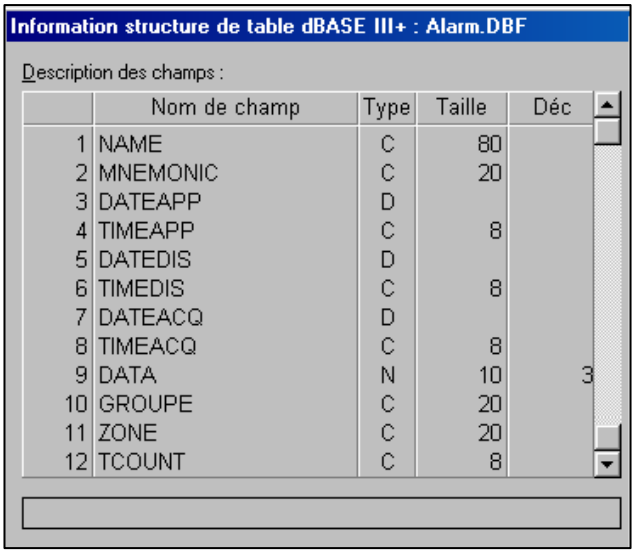
Serveur SMTP : Nom du serveur SMTP utilisé par le PC de votre application finale. Le serveur SMTP (fournisseur d'accès) permet d'envoyer des Emails en cas d'alarme.

Données techniques

Nom du fichier de paramétrage des alarmes : Alarm.oco
Nom de la table de la base de données de stockage des alarmes : Alarm.DBF
Table type Dbase 3+

Alarm.DBF, Structure de la table de stockage des alarmes.

C : Type de caractères (80, 20 caractères).
D : Type date (25/12/01 = 25 décembre 2001).
N : Numérique (3 = 3décimales).



	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	

Programmation sur les Alarmes

Le fonctionnement des applications en multipostes nécessite éventuellement la consultation des alarmes sur plusieurs postes. Cependant le stockage ne doit être fait que sur une seule base de données généralement le serveur. Afin d'interdire le stockage des données sur des postes clients, il est possible d'appeler la procédure **TASK_DB_ALARM** (**Stock** : Boolean); Si le booléen stock=true toutes les fonctionnalités paramétrées sont en service.
Si le booléen = False alors les fonctions Stockage, Email, Liaison série et klaxon sont dé validées.

Interdire le stockage générale des alarmes

```
SObject Procedure Circle1;
Begin
  Task_DB_Alarm(LeServeur); // Si leServeur=True les consignation sur disque,
End;                        // email, série & klaxon sont validées
```

Function TASK_NB_HISTOALARMS : **Integer**;
Cette fonction renvoie le nombre d'alarmes apparues et disparues depuis le début ou la remise à zéro avec la fonction TASK_KEEPHISTOALA.

Function TASK_GETHISTOALA (**Index** : Integer) : **String**;
Renvoie l'alarme à la position Index dans la liste des alarmes historisées.

Function TASK_KEEPHISTOALA (**NB** : Integer) : **Integer**;
Garde les X dernières Alarmes dans l'historique interne.

Procedure TASK_ACK_ALARM (**Alarm** : String);
Acquittement d'une Alarme par Index

Exemple :

```
// *** Récupère l'historique ********
For I:=0 to (Task_nb_Histoalarms-1) Do
 Begin
   TS:=Task_gethistoala(I);   //  Récupère Alarme
   Addstring(TS) ;            //  ajoute dans listbox
 End;
```

Appel de procédure personnalisée à l'apparition d'une alarme

Variable CallBackPrintAlarm de type pointer (Integer).

Cette variable permet de renseigner l'adresse d'une procédure appelée par le système à chaque apparition d'une nouvelle alarme.
Cette procédure peut effectuer un traitement particulier à chaque alarme. Ceci complémente la procedure de callback vue précédement pour envoie de SMS.

Traitement interne le paramètre value passé à la procédure personnalisée est formaté ainsi :

```
Value:=Date+' '+Heure+' '+ Intitule;
AppelCallBack(' '+Value+#13+#10);
```

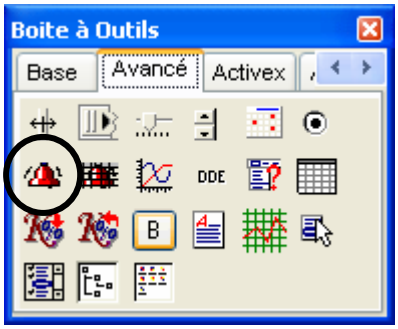
Exemple :

```
// *** Appel de ce programme par le gestionnaire d'alarmes **
Procedure MonProg ( Value : String);
Begin
  // *** Garde seulement l'intitulé
  While Pos(' ',Value)>0 do
  Begin
    Delete(Value,1, Pos(' ',Value));
  End;
  // *** enlève #13 & #10 (CR et LF)
  Value:=Copy(Value,1,Length(Value)-2);
  // *** Value contient ici le texte de l'alarme ****
  If Value='STOP' Then MW10:=MW10+1;
End;

// *** ListBox1 ************************************
SObject Procedure ListBox1;
Var
 PT : Integer;
Begin
  If Start_Visu Then
 Begin
   PT:=Addr(MonProg);
   CallBackPrintAlarm:=PT^;   // *** Renvoi vers MonProg **
 End;
End;
```

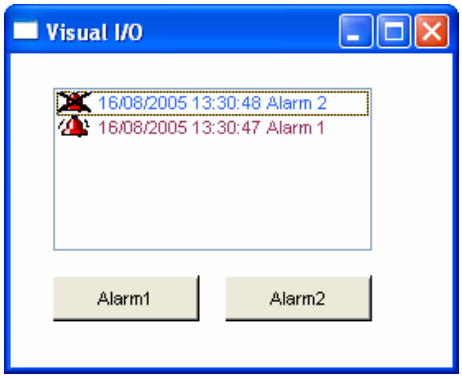
Affichage simple des alarmes

Composant Liste alarmes

Ce composant permet d'afficher les alarmes présentes.
C'est la forme la plus simple d'affichage des alarmes dans un synoptique.



Aspect en Run



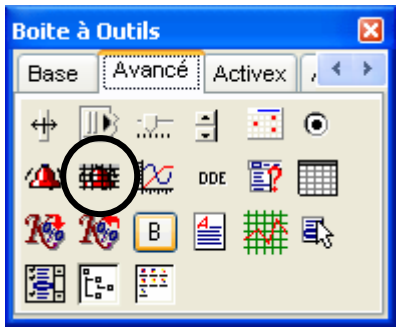
◄ Ici 2 alarmes sont affichées.
La première (Alarm 2) à disparue mais n'a pas été encore acquittée par l'opérateur.
La seconde (Alarm 1) est présente mais pas acquittée.

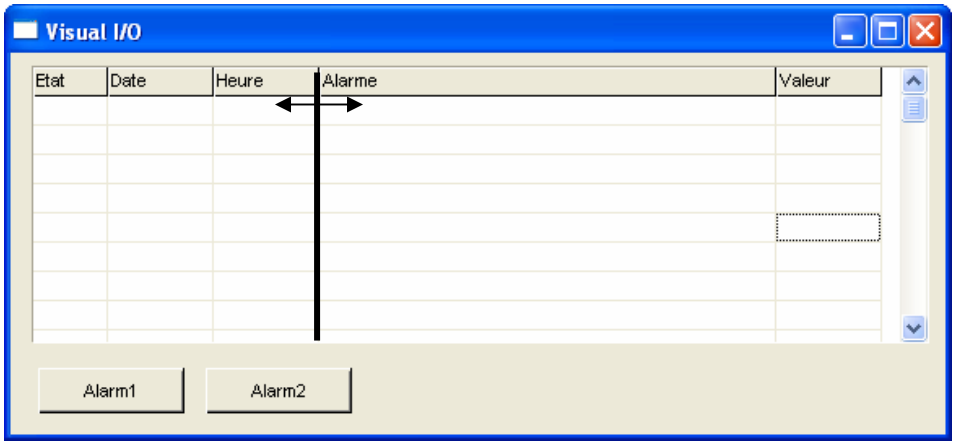
1. L'affichage de la cloche permet de signaler à l'opérateur que l'alarme est présente mais non acquittée.
2. L'affichage de la cloche barrée permet de signaler à l'opérateur qu'une alarme à été présente mais a disparue. L'opérateur doit alors l'acquitter pour faire disparaître le texte dans la ListBox.
3. Le texte de l'alarme sans Cloche devant indique que l'alarme est encore présente et quelle à été acquittée soit par un opérateur ou en automatique (voir paramétrage des alarmes).

Affichage avancé des alarmes

Composant Grille alarmes

Ce composant permet l'affichage des alarmes sous forme de grille. Cette grille personnalisable permet d'afficher les informations nécessaire concernant les alarmes présentes.
Le composant permet aussi d'afficher les alarmes relatives à un groupe et à une zone définie.
Enfin il permet de modifier ou supprimer les intitulés des colonnes de visualisation.

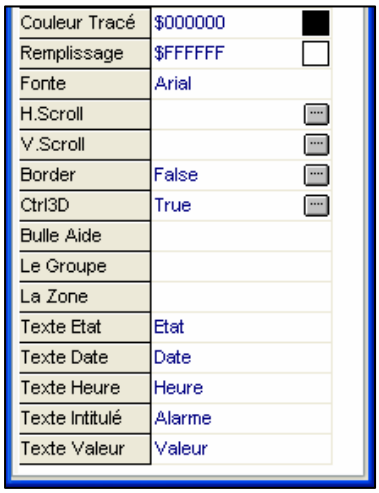




Lorsque le composant n'est pas sélectionné, vous pouvez régler la largeur des colonnes de la grille.

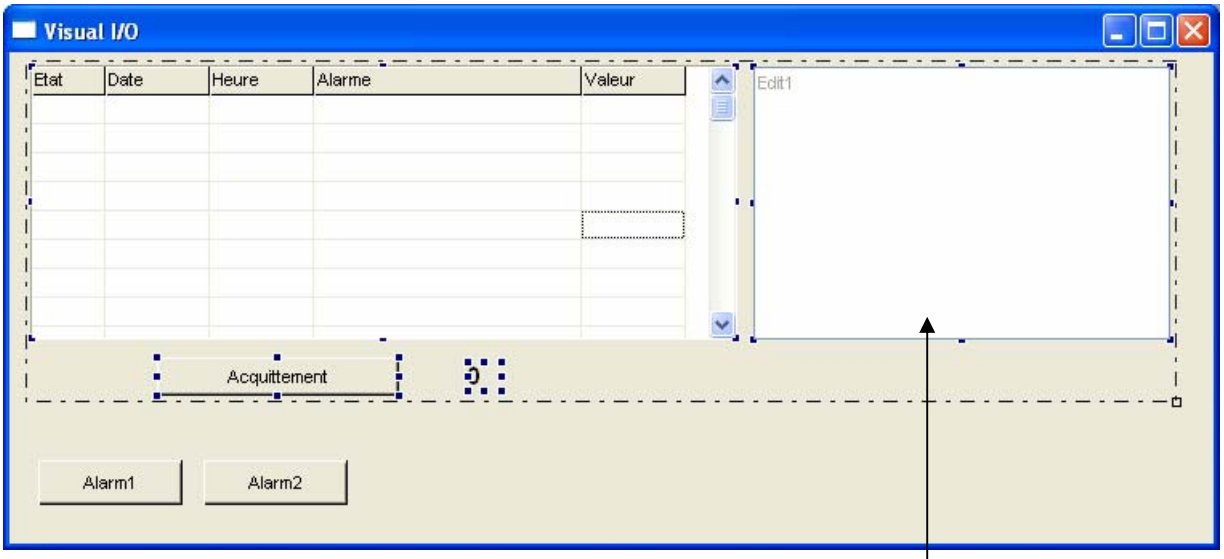
Ces largeurs sont sauvegardées dans le synoptique.

Propriétés graphiques du composant



Le Groupe : Si aucun nom de groupe n'est spécifié, les alarmes de tous les groupes apparaîtrons dans la grille.
La Zone : Si aucun nom de Zone n'est spécifié, les alarmes de toutes les Zones apparaîtrons dans la grille.
Texte Etat : Modifiable ou si aucun nom n'est spécifié, la colonne Etat ne sera absente.
Texte Date : Modifiable ou si aucun nom n'est spécifié, la colonne Date sera absente.
Texte Heure : Modifiable ou si aucun nom n'est spécifié, la colonne Heure sera absente.
Texte Intitulé : Modifiable ou si aucun nom n'est spécifié, la colonne Intitulé sera absente.
Texte Valeur : Modifiable ou si aucun nom n'est spécifié, la colonne Valeur sera absente.

Afficher le fichier d'aide de l'alarme

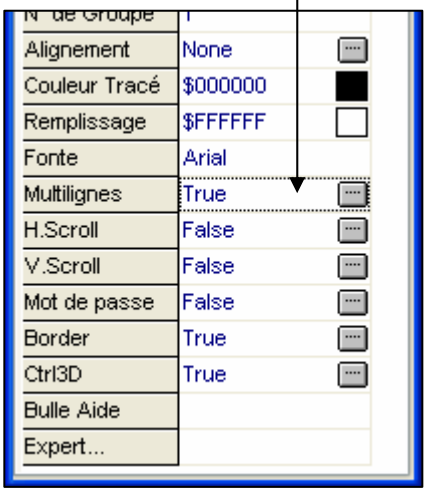


Si le composant est utilisé seul, la visualisation des alarmes est effectuée.
Si un composant **EditBox** est présent et est **groupé avec la grille**, le texte d'aide de l'alarme sélectionnée sera affichée automatiquement en Run. Vous devez régler les propriétés de l'EditBox en mode multilignes.

La grille d'alarmes peut afficher dans un champ de saisie Multi lignes le texte de l'aide. Si un bouton et une valeur numérique sont présents dans sa périphérie, la grille permet aussi l'affichage du nombre d'alarmes présentes ainsi que l'acquittement de toutes les alarmes de la grille.

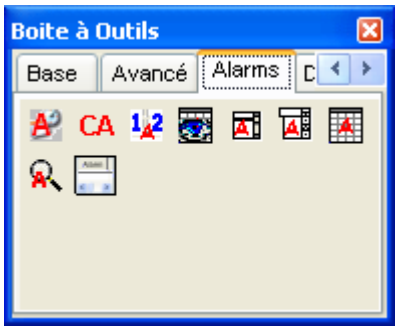
Placez un bouton et une valeur numérique, puis groupez ces éléments avec la grille.

En *Run* le bouton permettra d'acquitter en bloc toutes les alarmes présentes et la valeur numérique affichera le nombre d'alarmes présentes dans la grille.



Composants pour la gestion des alarmes

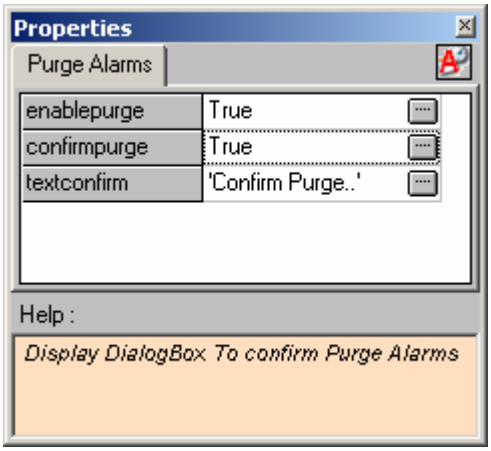
Ces composants sont livrés avec leurs sources, permettant d'effectuer des modifications ou des personnalisations.



De gauche à droite :

PurgeAlarms : Purger des alarmes
Cumul Alarm : Calcul de cumul sur une alarme.
Choice Date : Permet d'afficher un calendrier et choisir une date
Display Last Alarms : Affiche les dernières Alarmes.
Listbox Alarms List : Affiche les intitulés des alarmes.
Combobox Alarms List : Affiche les intitulés des alarmes.
Display Alarms To Date : Affiche les alarmes à une date précise.
Histo Alarms : Historique d'alarmes.
Alarms Extracting : Extraction d'alarmes

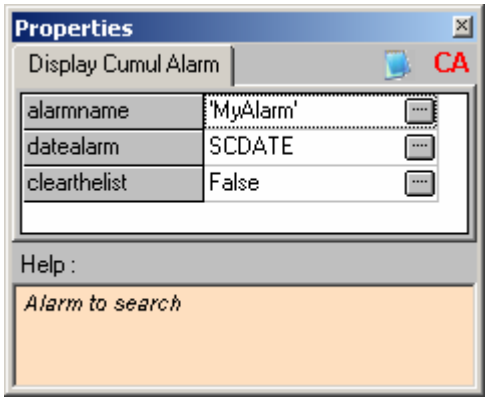
Purge Alarms



Composant permettant d'afficher un calendrier et de choisir une date de purge des alarmes

EnablePurge : Permet d'autoriser ou non l'utilisateur à faire une purge.
Confirmpurge : Quand l'utilisateur à sélectionné une date de purge, une boite de dialogue apparaît avec le texte de la cellule TextConfirm.
TextConfirm : Texte à afficher dans la boite de dialogue en cas de purge.
*Ce composant est moins nécessaire maintenant car **l'onglet configuration** des alarmes permet une purge automatique. Cette purge intervient quand le fichier de stockage atteint une taille prédéterminée.*

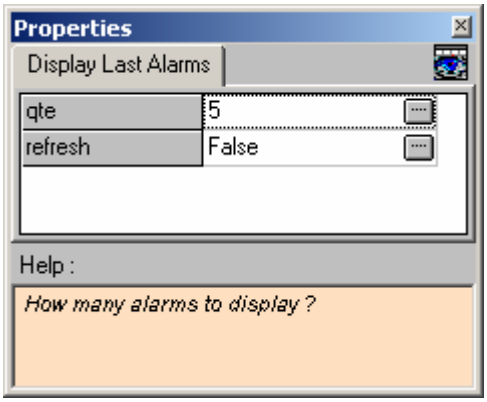
Cumul Alarms



Listbox permettant d'afficher le nombre d'alarmes à une date donnée.

AlarmName : Nom de l'alarme à recherchée.
DateAlarm : Date de l'alarme pour recherche.
ClearTheList : Efface entièrement la Listbox.

Display last alarms

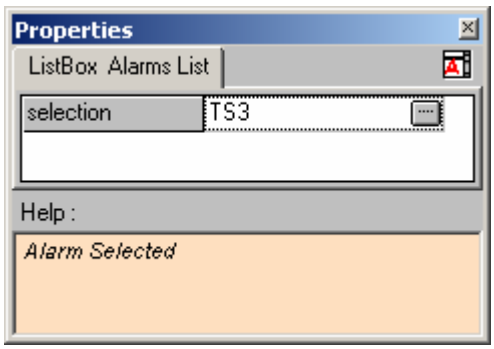


Composant permettant de visualiser les X dernières alarmes dans une grille .

Qte : Quantité à visualiser.
Refresh : Rafraîchi la grille des X dernières alarmes.

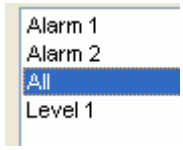
Note : Le passage de la souris permet aussi le rafraîchissement.

ListBox alarms list

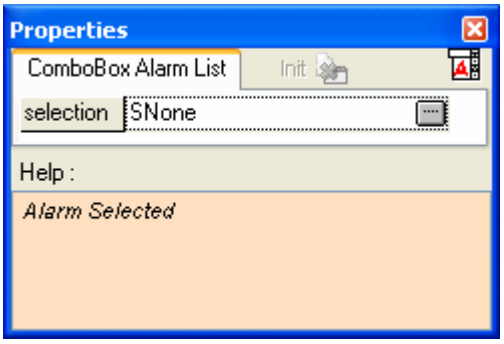


Listbox affichant les intitulés d'alarmes.

Selection : Variable Chaîne de caractères recevant l'alarme sélectionnée dans la liste.



Combobox alarms list

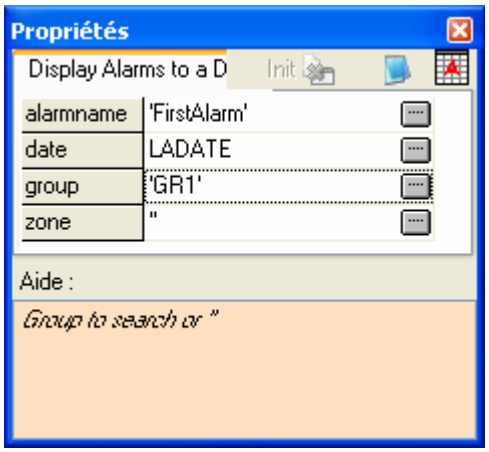


Combo box affichant les intitulés d'alarmes.

Selection : Variable Chaîne de caractères recevant l'alarme sélectionnée dans le combobox.

Tous les intitulés d'alarmes apparaissent.
Le texte **All** est ajouté par le système permettant ainsi une sélection globale de tous les intitulés.

Display alarm to a date

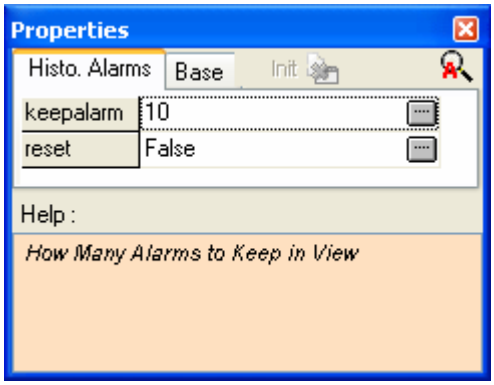


Composant permettant d'afficher les apparitions d'une alarme à une date donnée.

AlarmName : Nom de l'alarme à rechercher.

Date : Date de recherche.
Group : Group de tri (optionnel) ou chaîne vide.
Zone : Zone de tri (optionnel) ou chaîne vide.

Histo. alarms



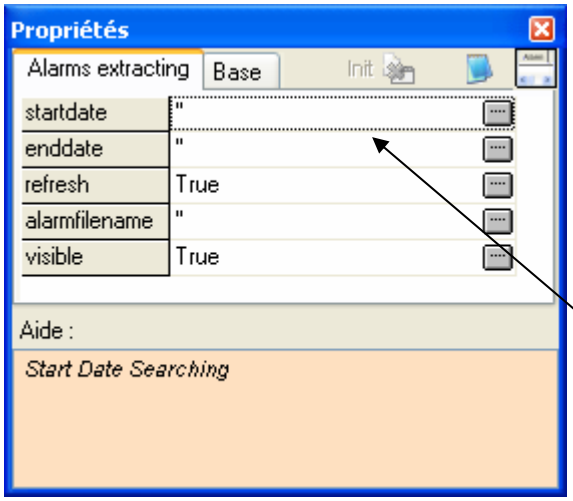
Composant permettant de visualiser dans une listbox toute les dernières alarmes survenues.

KeepAlarm : Quantité d'alarmes à visualiser.
Reset : Remise à zéro de la listbox.

Si keepAlarm = -1 alors toutes les alarmes sont gardées en mémoire.
Attention au débordement de mémoire.

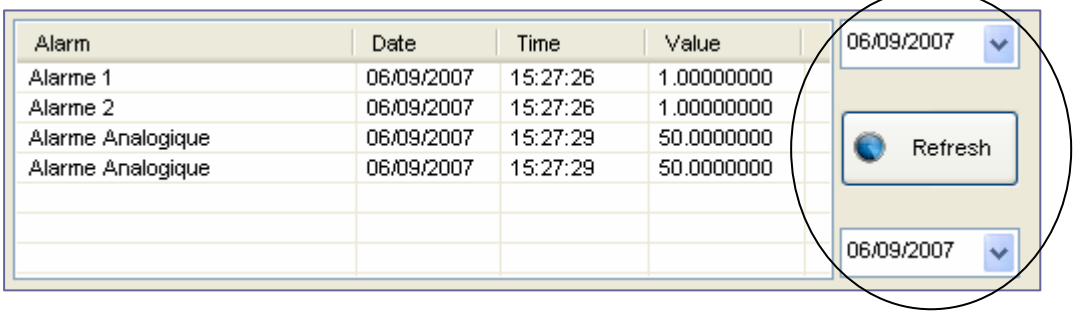
Alarm Extracting

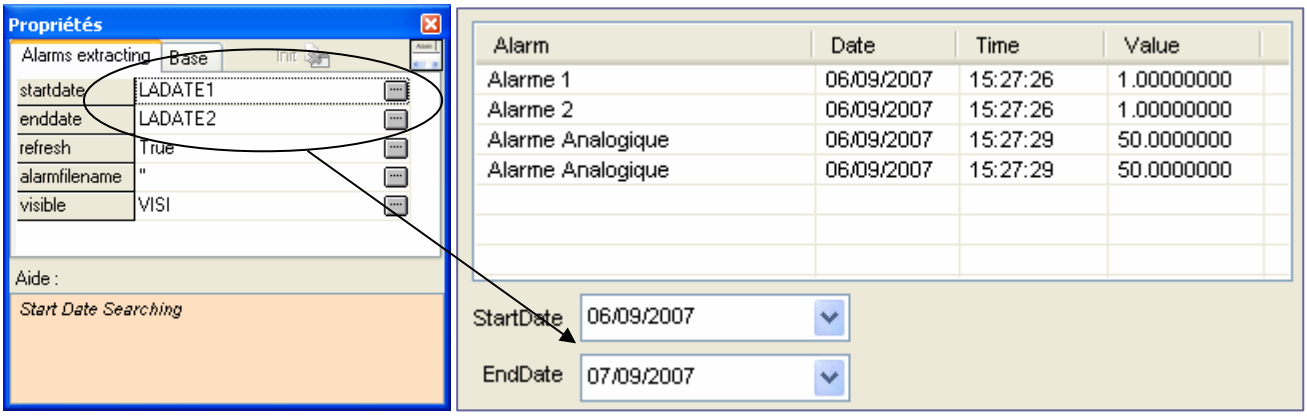
Ce composant permet d'afficher les alarmes apparues depuis une date (startdate) jusqu'à une autre date incluse (enddate). Si les 2 champs Startdate et enddate restent chaine vide (comme ci-dessous) alors les 2 combobox s'affichent sur le coté de la grille ainsi que le bouton refresh. Sinon ces 3 objets disparaissent au profit de 2 variables chaines contenant les 2 dates de recherche.



StartDate : Date de début de la recherche.
EndDate : Date de fin de la recherche.
Refresh : Commande auxiliaire de raffraichissement de la grille. Fonctionne en parallèle du bouton.
AlarmFilename : Si chaine vide le composant travaille avec Alarm.dbf sinon utilise le chemin (Ex : "C:\Myproj\Alarm.dbf")
Visible : Affichage disparition par bit du composant.

Ici Startdate et Enddate sont égales à chaines vides alors les 2 combobox s'affichent pour la sélection de ces dates.



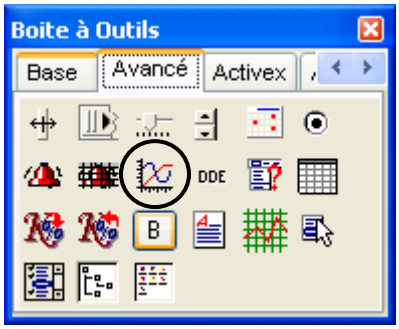


Dans ce cas de figure les 2 combobox ainsi que le bouton refresh de droite disparaissent. Ils sont remplacés par 2 variables chaines venant de l'application.

Tracés simples de courbes au fil du temps

Visual IO possède 2 composants permettant de tracer des courbes au fil du temps.
Les composants décrits ici ne permettent pas dans leurs paramétrages, l'enregistrement sur le disque des valeurs qu'ils affichent.

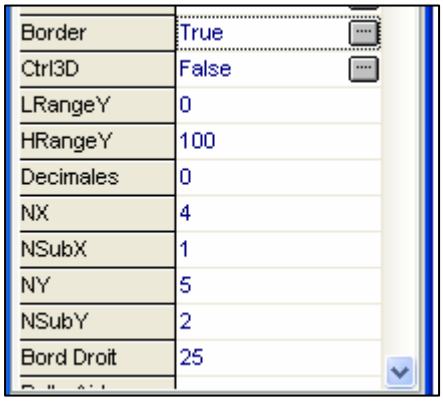
Premier Composant



Nom : Courbes

Ce composant permet l'affichage au fil du temps de 8 courbes simultanées.
Ce composant ne stocke aucune donnée. Le traçage des valeurs sont font au fil du temps, même si le synoptique n'est pas présent à l'écran.

Boite de propriétés

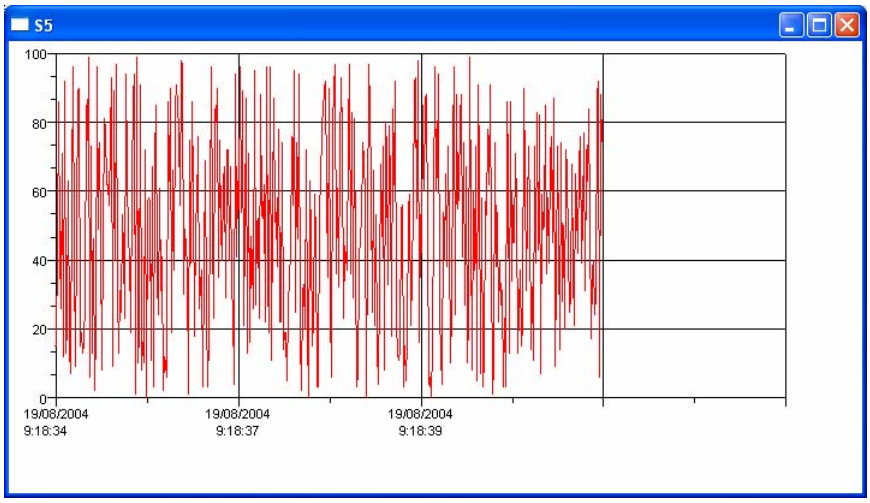


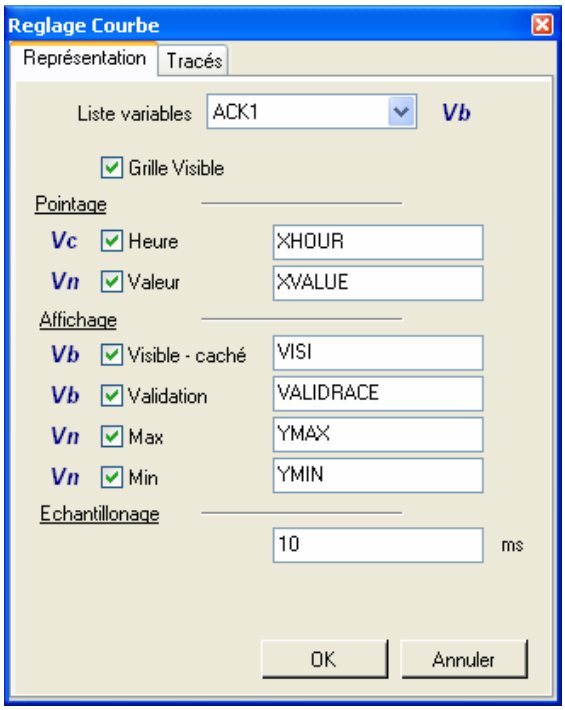
Border : permet d'encadrer la courbe ou non.

LRangeY, HRangeY : Règle le minimum et le maximum de l'axe Y.

NX, NSubX : Règle le nombre de divisions et de sub divisions de l'axe X.

Bord Droit : permet de régler l'espace entre le bout de l'axe X et le cadre graphique.





Grille visible : Permet d'afficher ou non une grille sur la courbe.
Heure : Variable chaîne permettant de récupérer l'heure pointée par la souris. Lorsque la souris se déplace sur la courbe, cette variable reçoit l'heure de tracé du point visé.
Valeur : Variable numérique récupérant la valeur du point pointé par la souris.

Visible – Caché : Permet en fonction de l'état d'une variable booléenne, d'afficher ou de cacher la courbe.
Validation : Permet en fonction de l'état d'une variable booléenne de valider ou de stopper le tracé de toutes les courbes au fil du temps.
Validation : Permet d'autoriser ou stop le traçage des courbes (Pause/résume).

Max : variable numérique permettant de spécifier la valeur maximum de l'axe Y.
Min : variable numérique permettant de spécifier la valeur minimum de l'axe Y.

Echantillonage : Constante numérique spécifiant en ms la fréquence d'affichage des points.

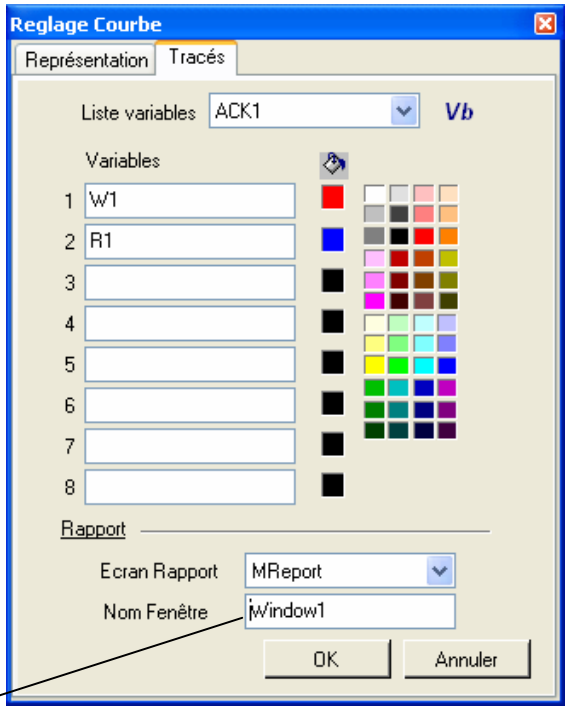
8 Stylos sont disponibles dans le visualisateur de courbes. Chaque stylo reçoit une variable numérique (integer, Real) ou une variable booléenne ainsi qu'une couleur.

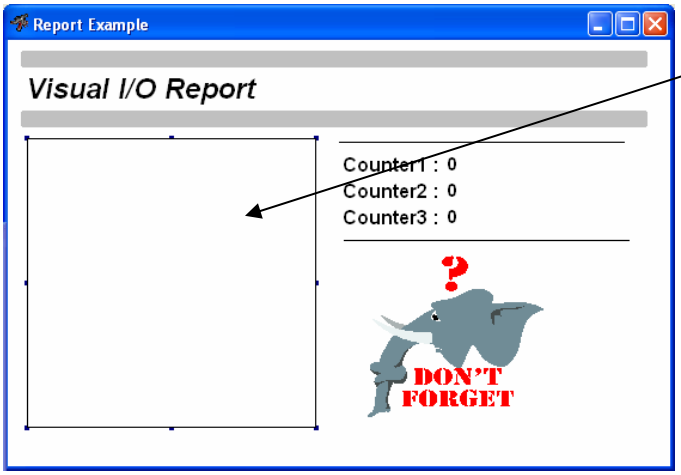
Rapport :

Visual I/O permet l'édition de rapports sophistiqués ou dans un cadre il est possible de recevoir une courbe. Ce cadre est tracé à l'aide de l'objet fenêtre dans un synoptique qui sert de modèle de rapport.

Au moment de l'édition, afin de remplacer ce cadre fenêtre par le tracé de la courbe, spécifiez dans nom fenêtre le nom de la fenêtre du rapport. Spécifiez aussi le nom de synoptique servant de modèle de rapport.

Au lancement du rapport par l'instruction **PrintReport**, le cadre de fenêtre sera remplacé par le visualisateur de courbes en cours.

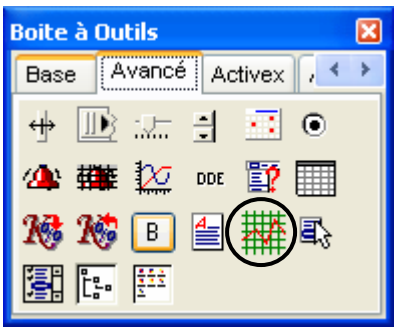




Le synoptique ci-contre est tracé tout à fait normalement, mais il ne sera pas affiché à l'écran.
Il sera imprimé et le cadre de fenêtre sera remplacé par les courbes présentes au moment de l'exécution de l'instruction **PrintReport**.

Ici le synoptique s'appelle MReport et la fenêtre Window1.

Deuxième Composant

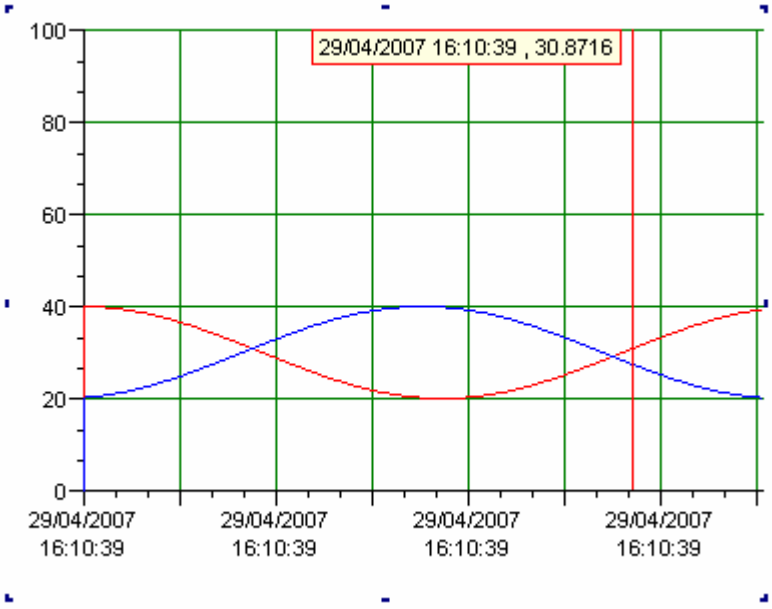


Nom : TrendHisto

Ce composant permet l'affichage au fil du temps de 4 courbes simultanées.
Ce composant ne stocke aucune donnée. Le traçage des valeurs sont font au fil du temps même si le synoptique n'est pas présent à l'écran. Ce composant inclus un buffer tournant permettant un défilement avant arrière des valeurs internes enregistrées.

Boite de propriétés

Bulle Aide	
Grille...	True
NX	6
NSubX	2
NY	5
NSubY	2
Couleur Axes...	0
Couleur P1...	255
Couleur P2...	16711680
Couleur P3...	32768
Couleur P4...	128
YMax	100
YMin	0
Décimales	4
Larg. Stylo Impr.	2
Var1	W0
Var2	Temperature
Var3	Pression
Var4	Compteur
Echantillonage X10ms	10



Grille Affiche ou non la grille. La couleur de la grille est fixée par la couleur de tracé.

NX,NSubX,NY,NSubY : Nombre de graduations principales et graduations secondaires sur les axes X et Y.

Couleur P1, P2, P3, P4 Couleurs des stylos de traçage des courbes.

YMax et Ymin Minimum et maximum de l'axe Y.

Decimales : Nombre maximale de décimales à afficher sur les textes de l'axe Y.

Larg. Stylo Impr. Largeur du stylo en cas d'impression des courbes.

Var1 à Var4 : (optionnel) Nom des variables à afficher au fil de l'eau par le composant, si celui-ci doit être utilisé en traceur.

Echantillonage : Valeur à multiplier par x10ms pour l'échantillonnage et l'affichage de nouvelles valeurs (Var1 à Var4).

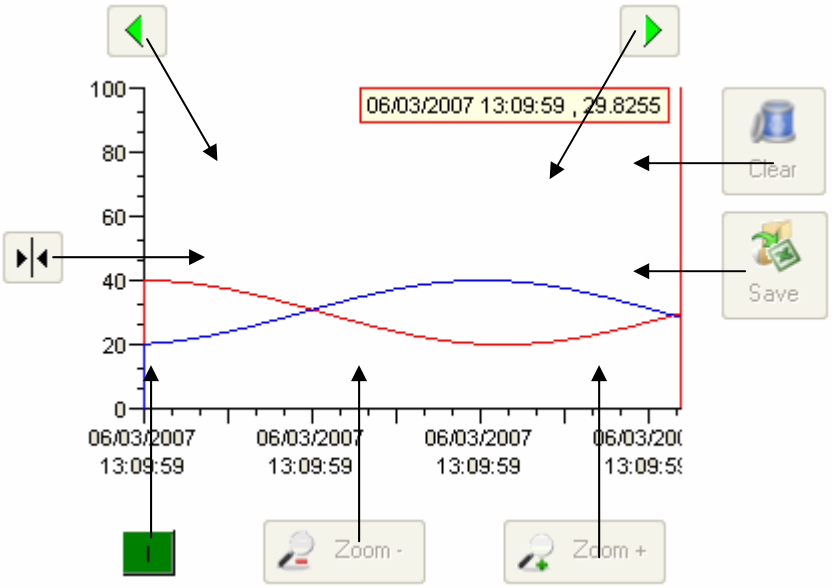
Note : La modification des attributs de la courbe est possible en utilisant les composants de l'onglet Trends

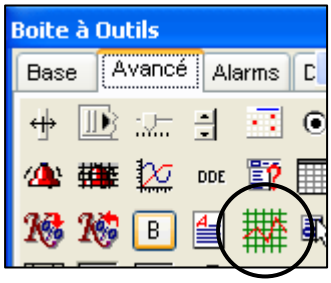
Création d'une page de visualisation de courbes

Les composants décrits ci-dessous servirons aussi bien pour naviguer sur une courbe au fil du temps ou sur un historique.

Composants de l'onglet Trends

Cet onglet permet à l'aide de composants périphériques d'enrichir la courbe de base que vous trouverez dans l'onglet Avancé. Si une seule courbe est utilisée dans votre synoptique, tous les composants fonctionneront avec celle-ci.
Si plusieurs courbes doivent être utilisées, il faudra grouper les composants avec la courbe qui est associée.

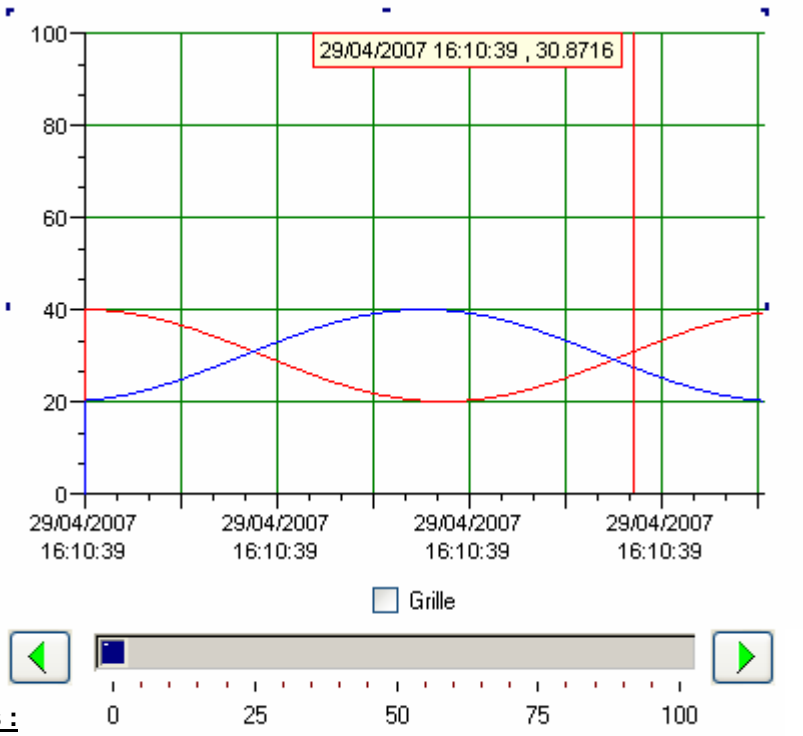




Placez un traceur de courbe, puis tous les composants en périphérie. Ceux-ci fonctionneront automatiquement avec ce traceur.

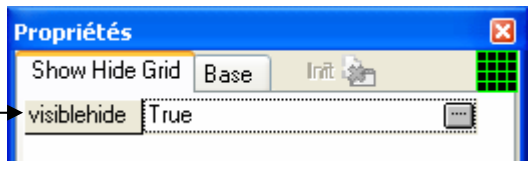
Navigation dans les courbes

Placez une courbe (composant TrendHisto de l'onglet Avancé).



Paramètres communs des composants :

VisibleHide est commun à tous les composants. Ce paramètre du type booléen (bit) permet d'afficher ou de cacher le composant sur le synoptique. Ceci permet de simplifier ou d'enrichir à un moment précis l'affichage général d'un synoptique.



Affichage d'une grille 

Ce composant permet à l'aide d'une case à cocher, d'afficher ou de cacher la grille sur le traceur de courbes.

Aller au premier point 

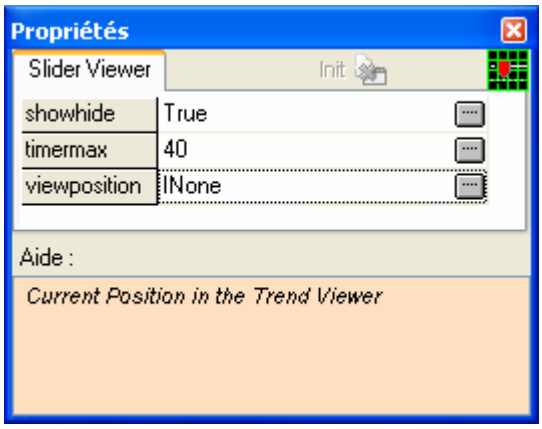
Ce composant permet de repositionner le traceur de courbes sur le premier point enregistré.

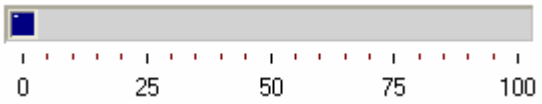
Aller au dernier point 

Ce composant permet de repositionner le traceur de courbes sur le dernier point enregistré
Ce composant permet aussi de repasser le traceur de courbes en affichage au fil de l'eau.

Navigation dans le traceur de courbes. 

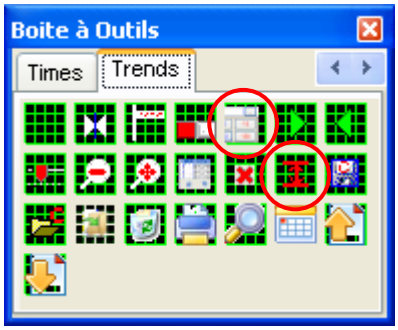
Ce composant permet naviguer dans le traceur courant. Le traceur intègre en interne un buffer permettant de stocker un nombre important de points supérieur au nombre visible.
Chaque canal (courbe) possède 8400 points stockable dans un FIFO.

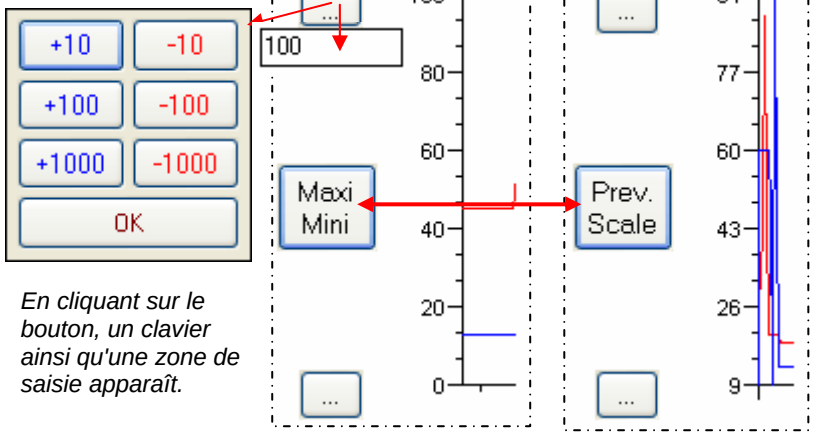




TimerMax : en 1/10sec temps permettant de rendre le focus à windows si plus aucune action n'est effectué sur le potentiomètre. Rendre le focus (sélection) permet au système de rafraîchi le curseur en fonction des nouveaux enregistrements dans le traceur de courbes.
Viewposition : Position courante dans le traceur de courbes. Report éventuel dans une variable Integer.

Graduations de l'axe Y



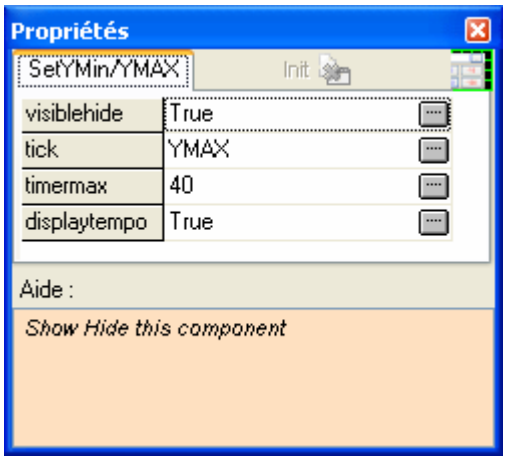


En cliquant sur le bouton, un clavier ainsi qu'une zone de saisie apparaît.

En cliquant sur le bouton **Maxi/Mini** *L'échelle s'auto ajuste au maxi et au mini des courbes visualisées. En cliquant de nouveau sur le bouton, l'opérateur revient aux anciennes valeurs de l'axe Y.*

Fixer le maximum ou le minimum sur l'axe Y

Ce composant permet d'afficher un champ de saisie ainsi qu'une boite à boutons pour modifier la valeur mini ou maxi de l'axe Y.

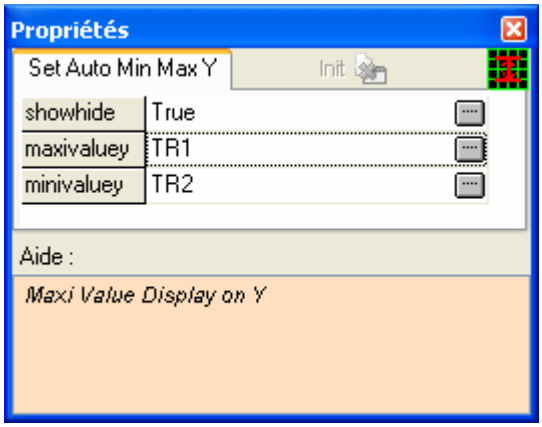


Tick : YMAX ou YMIN Permet de définir la graduation la plus haute ou la plus basse de l'axe Y
TimerMax : Temps maximum avant disparition du clavier numérique.
Displaytempo : True alors le temps restant avant la disparition du clavier numérique s'affiche sur le bouton. C'est simplement effet graphique.

La zone de saisie ainsi que le petit clavier se referme automatiquement à la fin de la tempo

Fixer automatiquement les valeurs Maxi et le Mini sur l'axe Y.

Ce composant permet de calculer et de fixer automatiquement les 2 valeurs (Maxi et Mini) de l'axe Y.



MaxiValueY & MiniValueY : Variables recevant les valeurs maxi et mini affectées à l'axe Y par un clic sur le bouton.

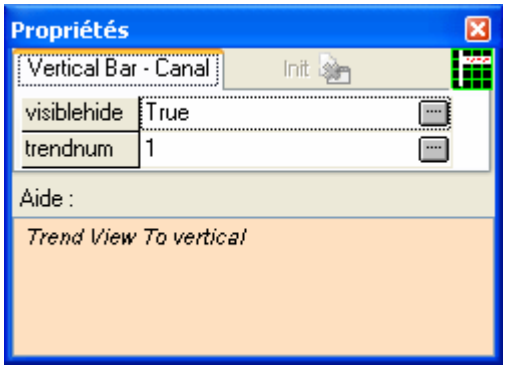
Curseur et couleurs des courbes

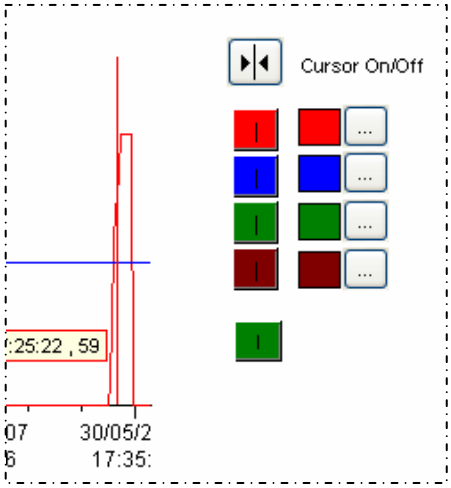
Affichage du curseur Vertical

Ce composant sous forme d'un bouton, permet d'afficher ou de cacher le curseur vertical qui est déplaçable à la souris et qui indique la valeur et la date du point sur lequel il se situe.

Affectation de la courbe au curseur Vertical

Ce composant permet d'affecter une courbe au curseur vertical. Ainsi déplacer le curseur permettra d'afficher uniquement les valeurs de cette courbe. L'axe X sera aussi gradué en temps ou en valeur numérique par rapport à la courbe choisie.

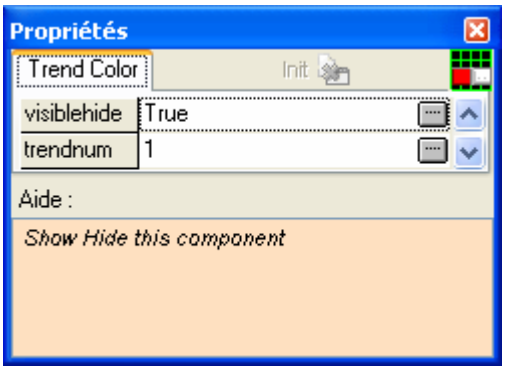




Trendnum : Permet de définir le numéro de courbe affecté au curseur vertical. Si vous désirez avoir un choix sur les 4 courbes, il faudra placer 4 composants avec le paramètre **TrendNum** différent.

Changement de la couleur d'une courbe

Ce composant affiche une palette de couleurs permettant de réaffecter une nouvelle couleur à une des courbes.



Trendnum : Permet de définir le numéro de courbe sur laquelle le changement de couleur doit s'effectuer.
Entrez une valeur numérique ou une variable de type integer.

Arrêt / Marche des enregistrements au fil de l'eau.

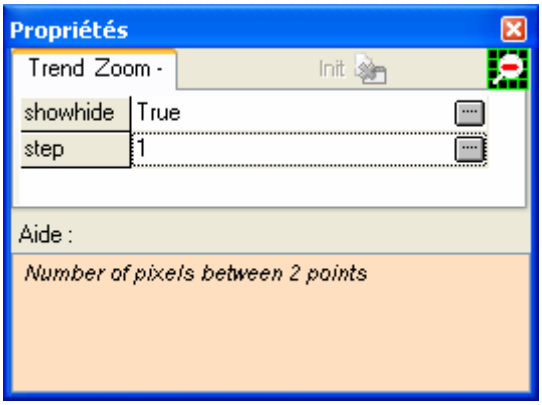
Ce composant permet d'arrêter ou de remettre en marche l'enregistrement des variables (4 maximums) paramétrées dans le traceur de courbes.

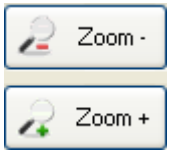
Zooms et multiplicateurs

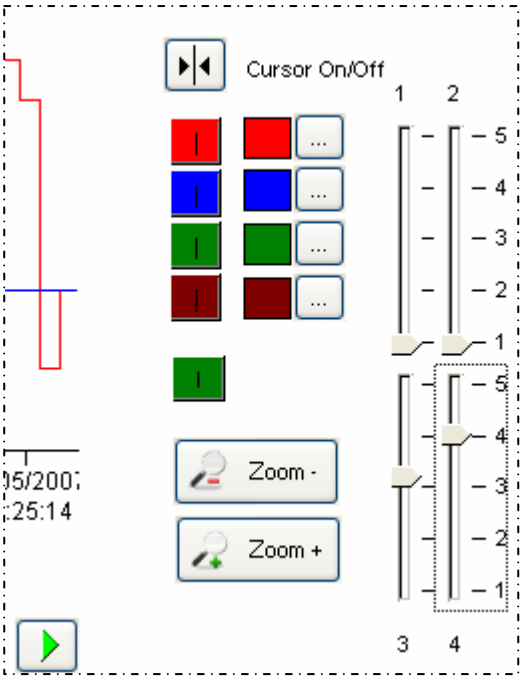
Ces composants permettent d'effectuer une expansion des courbes afin d'afficher tous les détails en X.
Un autre composant permet d'appliquer un Cœfficient a chaque courbe pour mieux les mêmes en correspondances ou coïncidences.

Zoom Plus et Zoom Moins.

Ces composants permettent de zoom c'est-à-dire d'agrandir ou de réduire l'espace entre 2 points enregistrés



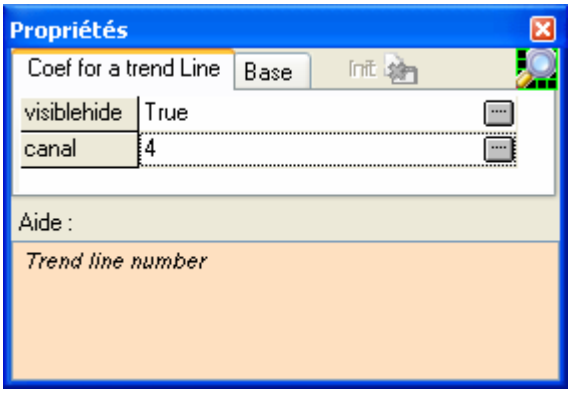




Step : Permet de définir un incrément ou un décrément pour l'espacement des entre les points a chaque click sur le bouton Zoom Plus ou Zoom moins.
Avec Zoom Plus à chaque click l'écartement entre les points est incrémenté de la valeur de Step.
L'inverse est effectué pour zoom Moins.

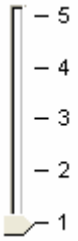
Application d'un Cœfficient aux valeurs d'une courbe.

Ce composant permet de multiplier toutes les valeurs d'une courbe avec un cœfficient.



Canal : Numéro de courbe affectée par le cœfficient.

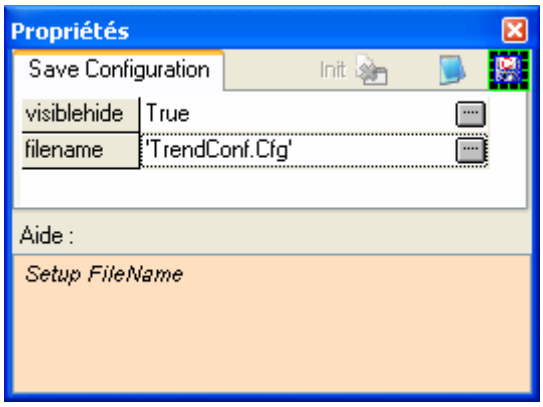
Dans l'onglet Base de ce composant, vous pourrez modifier tous les attributs du curseur. Tel que traçage en Horizontal ou Vertical, nombre de graduations, différentes couleurs etc.



Configurations

Sauvegarde de la configuration courante.

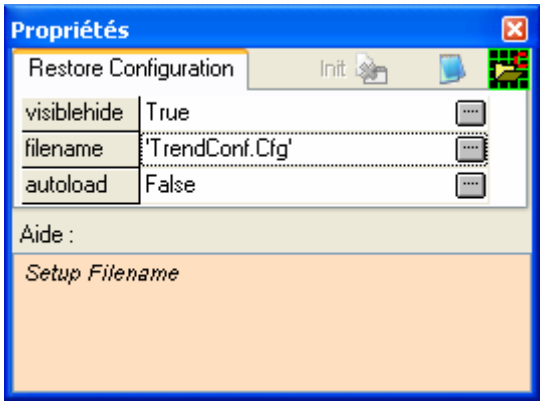
Ce composant permet de sauvegarder la configuration courante du traceur de courbes.
Sont sauvegardés zoom, couleurs des courbes, affectation du curseur vertical, maxi et mini de l'axe Y.



Filename : Nom du fichier de sauvegarde de la configuration courante.

Restauration d'une configuration.

Ce composant permet de sauvegarder la configuration courante du traceur de courbes.

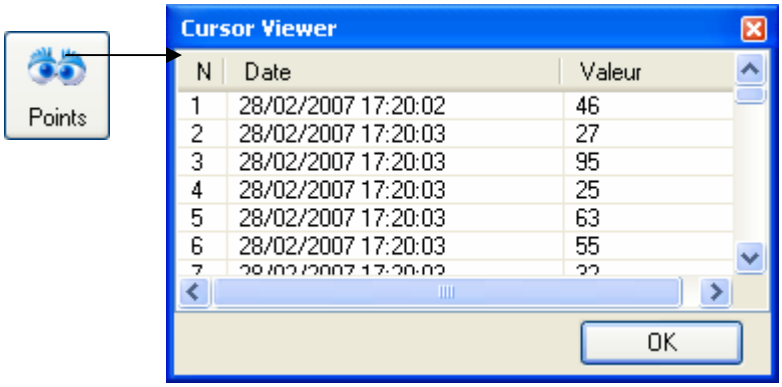


Filename : Nom du fichier de la dernière configuration courante sauvegardée.
Autoload : Si true la configuration est chargée automatiquement au premier affichage du synoptique.

Affichage, Sauvegarde et Chargement simples de points des courbes

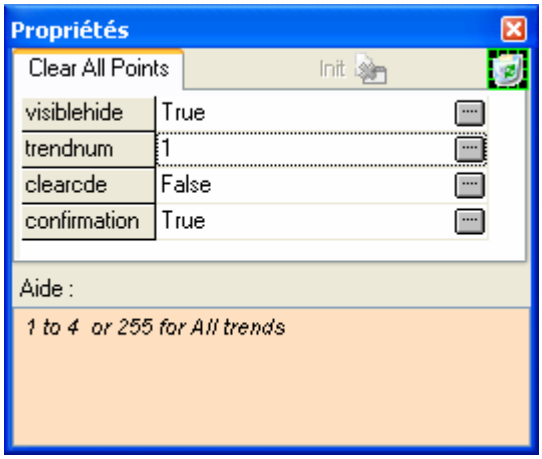
Affichage des points à partir du curseur.

Ce composant permet de faire apparaître une petite fenêtre affichant dans une grille les points à partir du curseur.



Effacement de tous les points d'une courbe.

Ce composant permet d'effacer tous les points d'une courbe précise enregistrés dans le traceur. Si le paramètre "**TrendNum**" est égal à 255 toutes les courbes sont effacées.



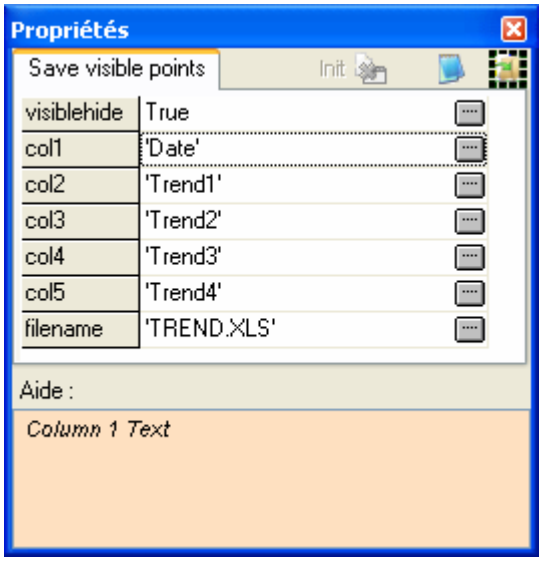
TrendNum : Numéro de la courbe à effacer (1 à 4). Si la valeur 255 est affectée a ce paramètre, toutes les courbes sont effacées.

ClearCde : Commande parallèle auxiliaire d'effacement.

Confirmation : Si true alors une boite de dialogue est affichée avant l'effacement des points. Si false aucune boite de dialogue n'est affichée.

Sauver les points visibles dans un fichier excel.

Ce composant permet de sauvegarder tous les points visibles dans un fichier compatible Excel. Le fichier est constitué de 5 Colonnes dont la plus à gauche contient toutes les dates.



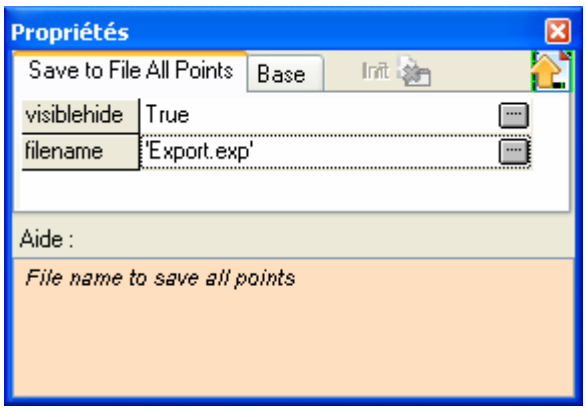
Col1 à Col5 : Texte de chaque colonne dans le fichier Excel généré.
FileName : Nom du fichier Excel généré.

Les dates stockées dans le fichier sont celles affichées sur l'axe X.

Ce composant ne sauve que les points des courbes visibles à l'écran.

Exporter tous les points vers un fichier.

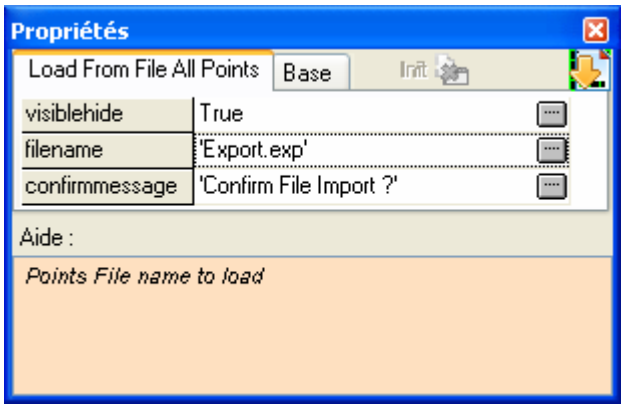
Ce composant permet de sauvegarder tous les points enregistrés dans le buffer interne du visionneur de courbes. Le fichier constitué est un fichier binaire au format propriétaire. Ce fichier ne peut être lu qu'avec l'autre composant "Importer les points d'un fichier" décrit ci-après.



FileName : Nom du fichier de sauvegarde des points.

Importer les points d'un fichier.

Ce composant permet d'importer dans les courbes les points contenus dans le fichier spécifié.
Avant l'importation des points toutes les courbes existantes sont remises à zéro c'est-à-dire que le visionneur est alors vidé.



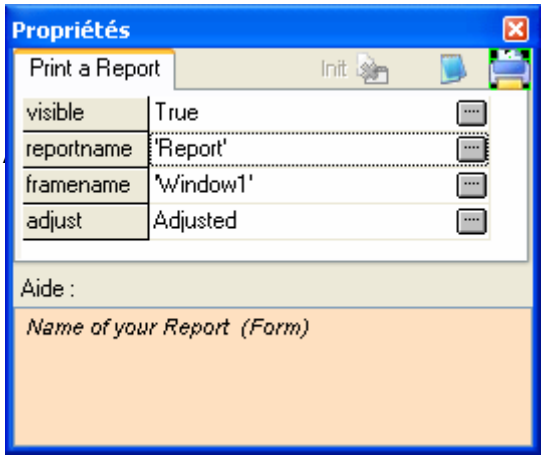
FileName : Nom du fichier contenant les points.

ConfirmMessage : Texte apparaissant dans une boite de message de confirmation avant d'importer les points dans les courbes. Si le message paramétré est vide " alors aucune boite de confirmation n'apparaît et l'import est effectué immédiatement.

Impression des courbes

Impression d'un rapport incluant un traceur de courbe.

Ce composant permet d'imprimer un rapport Visual I/O (voir chapitre) incluant un traceur de courbe. Le cadre de courbe peut fidèlement restituer ce qui est affiché dans le synoptique (adjusted) ou utiliser tous les points pour afficher une courbe beaucoup plus longue que celle affichée sur le synoptique (l'imprimante ayant plus de définition que votre écran).

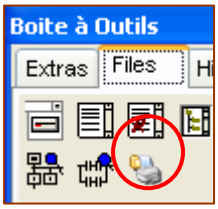


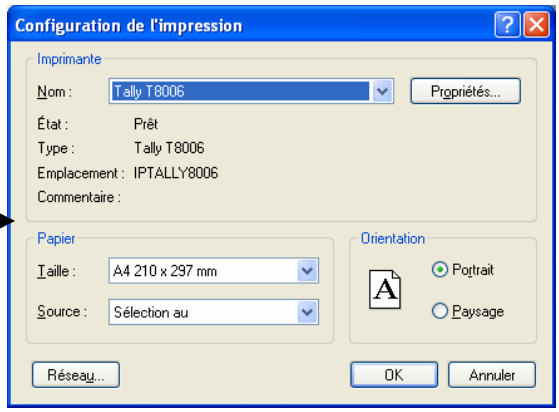
ReportName : Nom du rapport incluant le cadre délimitant le traceur de courbe.
Framename : Nom du composant fenêtre délimitant le traceur de courbe dans le rapport.
Adjust :
Adjusted : Reproduit le tracé du synoptique en espaçant les points sur l'imprimante.
Large : Trace les points l'un à coté de l'autre donnant ainsi une largeur de courbe beaucoup plus importante que celle affichée sur le synoptique.

Se référer au chapitre rapport pour la mise en œuvre de rapport dans Visual I/O

Choisir et régler l'imprimante

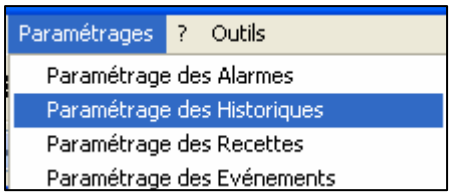
Ce composant permet de changer l'imprimante en cours et de régler ses différents attributs. Ce composant se trouve dans l'onglet Files



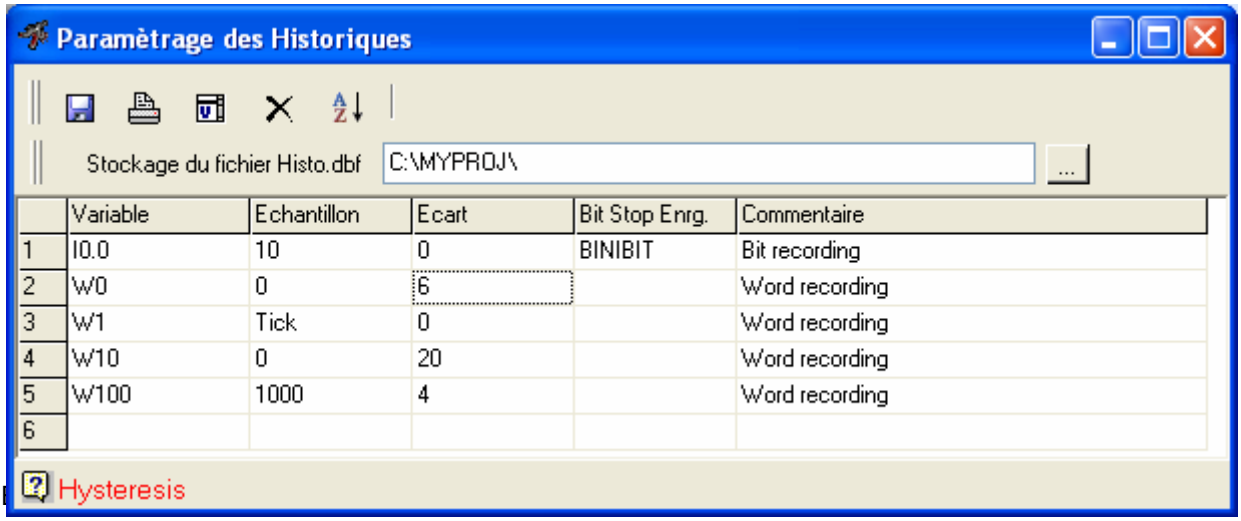


Historiques

Paramètrages



Une grille de paramétrage des historiques apparaît permettant d'historiser des variables booléennes ou numérique (integer, real ..). En cliquant sur le bouton sauver, Visual I/O génère et compile automatiquement un fichier source nommé **PCHISTO.PAS**. Ce fichier est consultable (et modifiable) dans la rubrique programme du navigateur de l'éditeur.



	Variable	Echantillon	Ecart	Bit Stop Enrg.	Commentaire
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					

Ce fichier compilé **PCHISTO.VPU** est intégré automatiquement dans la liste *Constitution de l'application/ Taches prioritaires* dans le menu **Utilitaires** de l'éditeur.

Variable : Variable Bit ou mot à historiser.
Echantillon : Période exprimée en ms (10 ms minimum) ou si=0 l'enregistrement se fait sur changement de valeur.
Ecart : Ecart significatif par rapport au dernier stockage pour de nouveau stocker la variable.
Bit Stop Enrg. : Bit facultatif permettant de stopper l'enregistrement de la variable.
Commentaire : Commentaire interne optionnel.

L'historisation peut être effectuée selon les différents cas de figure symbolisés et illustrés ci-dessous :

Cas	Echantillon	Ecart
1	0	0
2	0	10
3	10	0
4	10	10

L'enregistrement se fait

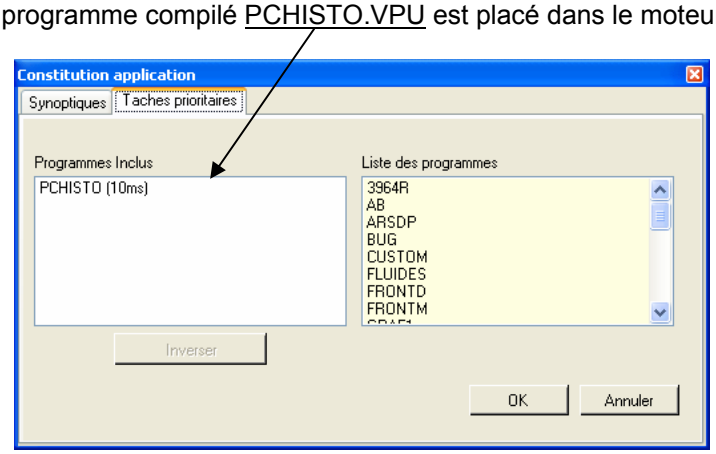
Cas 1 : Uniquement sur changement de valeur.
Cas 2 : Sur changement de valeur **ET** si celle-ci à changée au moins de 10 points.
Cas 3 : L'enregistrement se fait cycliquement tous les 100ms uniquement (Base de temps fixe).
Cas 4 : L'enregistrement se fait cycliquement tous les 100ms **OU** si la valeur change dans un intervalle de 10 points par rapport à sa précédente valeur. (Ce cas est intéressant pour limiter la taille de la base de donnée des historiques).

Autre cas ici :
La variable **I0.0** est historisée tout les 10 ms si le bit Binibit est à False.

La grille de paramétrage permet de :

Décrire les historisations de variables.
De créer un fichier Histo.oco (données du paramétrage).
De compiler un programme nommé PCHISTO.VPU, qui sera appelé cycliquement par le système.

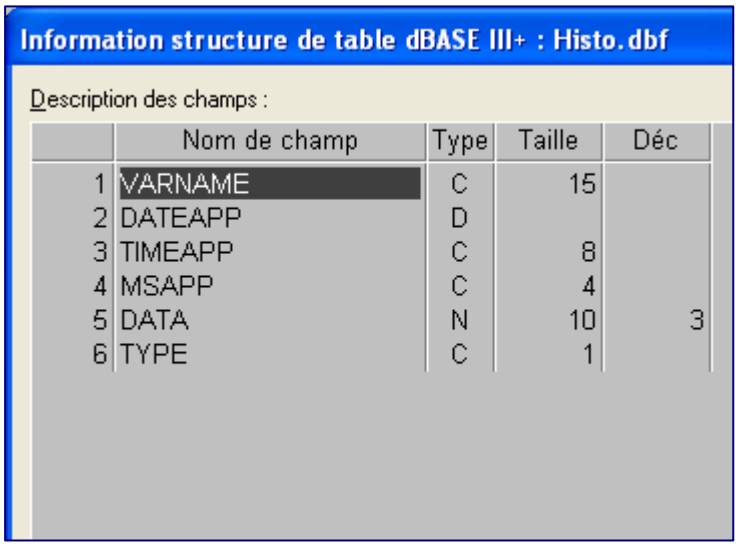
Après compilation le programme compilé PCHISTO.VPU est placé dans le moteur temps réel de Visual I/O.



Données techniques

Nom du fichier de paramétrage des alarmes : Histo.oco
Nom de la table de la base de données de stockage des alarmes : Histo.DBF
Table type Dbase 3+

Histo.DBF, Structure de la table de stockage des historiques.



	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	

VarName : Champ caractère contenant le nom de la variable historisée 15 caractères Maxi.
DateAPP : Date de stockage de la variable.
TimeAPP : Heure de stockage de la variable.
MSAPP : Millisecondes de stockage de la variable.
DATA : Valeur de la variable stockée. Avec 3 décimales maximum
TYPE : **B** = Booléen ou **V**=Valeur Numérique.

Note Le fichier des historiques est directement compatible avec **Excel**.

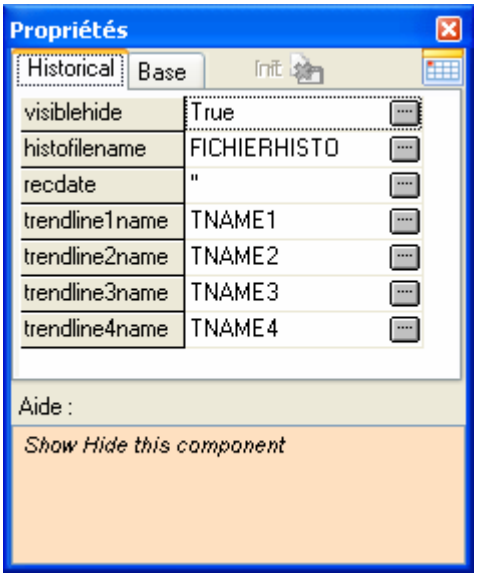
Affichage des historiques dans le traceur de courbes

Introduction

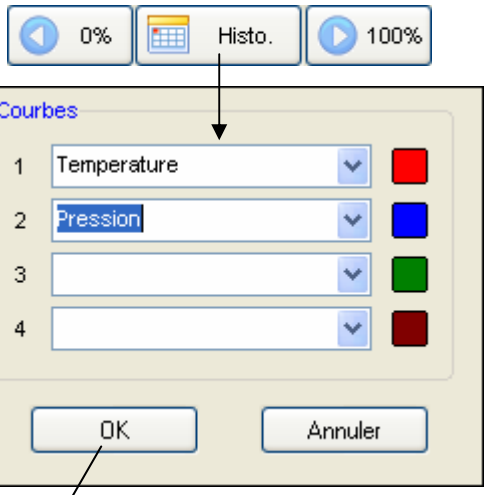
Comme vu précédemment le traceur de courbes sera utilisé avec les composants de l'onglet trends mais en visualisateur de points historisés par le système.

Visualisation d'un fichier historique.

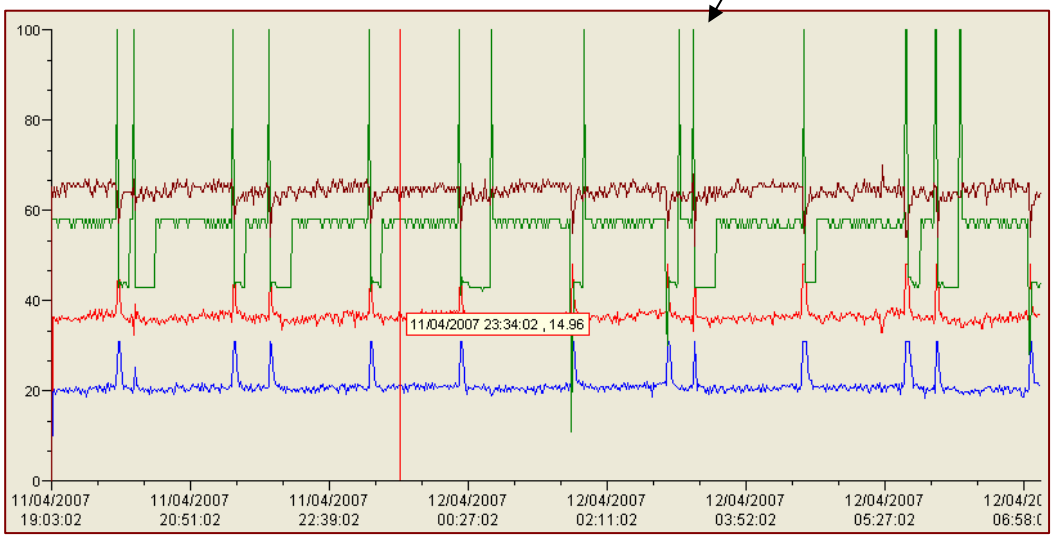
Ce composant permet l'affichage de l'historique de 4 variables spécifiées dont les valeurs sont contenues dans le fichier spécifié par le paramètre HistoFileName.



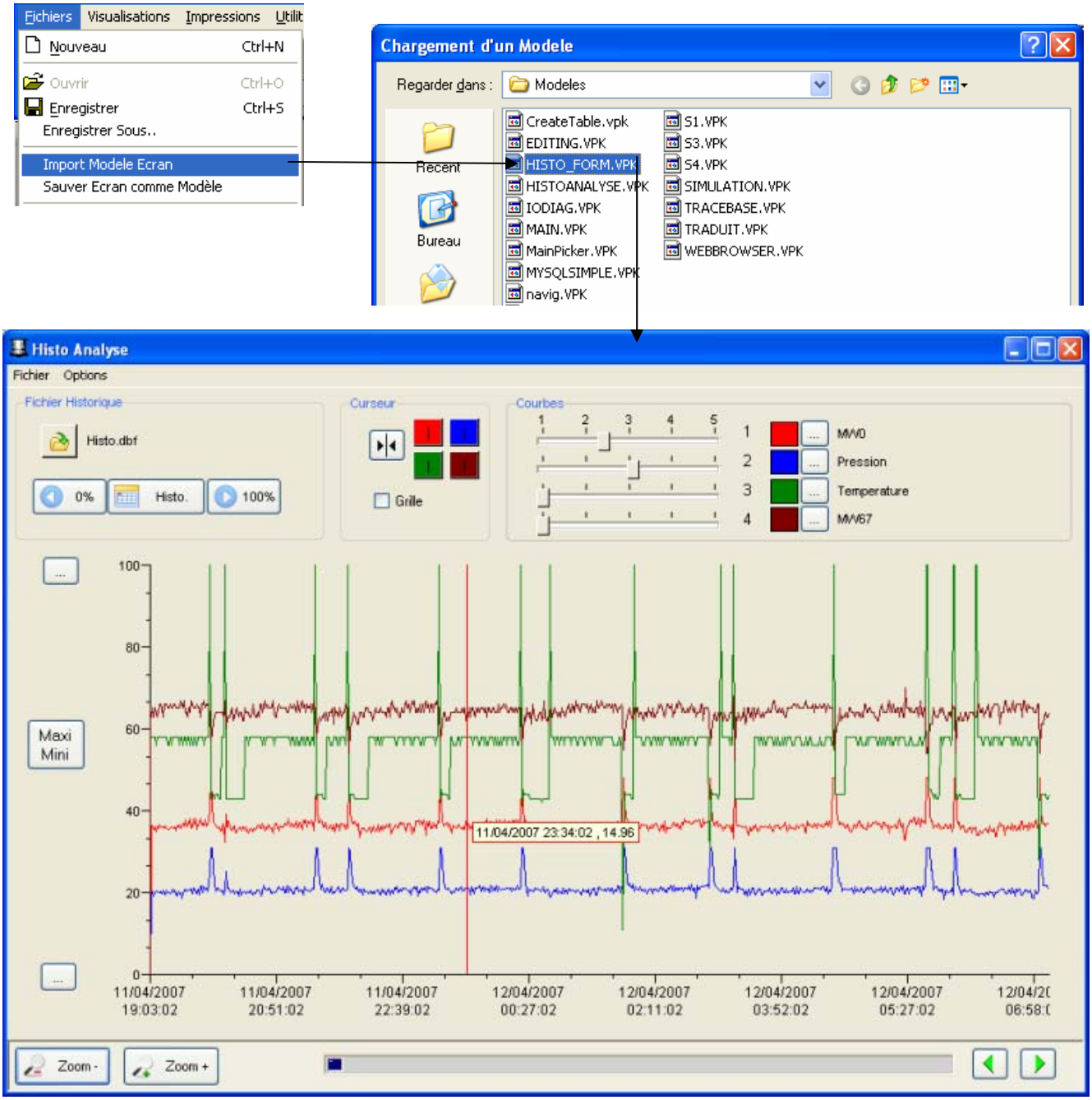
HistoFileName : Nom du fichier historique sur lequel doit s'effectuer la recherche. Si chaîne vide alors la recherche se fait sur le fichier Historique courant.
Recdate : Date de début de recherche ou chaîne vide la recherche commence au début du fichier historique.
Trendline1Name à Trendline4Name : Report vers des variables chaînes recevant le nom des variables choisies par l'opérateur.



En cliquant sur le bouton histo. L'opérateur peut alors choisir les 4 noms de variables pour le tracé des courbes.
En cliquant les boutons gauche et droite l'opérateur peut alors naviguer dans le fichier historique si celui-ci est important.
Le bouton gauche indique la position de début visible par le visualisateur et le bouton de droite indique la position de fin dans le fichier histo. Si les valeurs sont 0% (bouton gauche) et 100% (bouton droit) alors le fichier est complètement visible sur l'écran



Modèle d'écran de gestion des historiques prêt à l'emploi



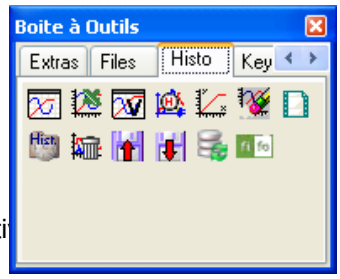
Le modèle HISTO_FORM permet la consultation des historiques. A incorporer directement dans votre projet sans déclaration de nouvelles variables globales.

Note : Pour l'impression voir chapitre impression des courbes.

Composants pour la gestion des historiques

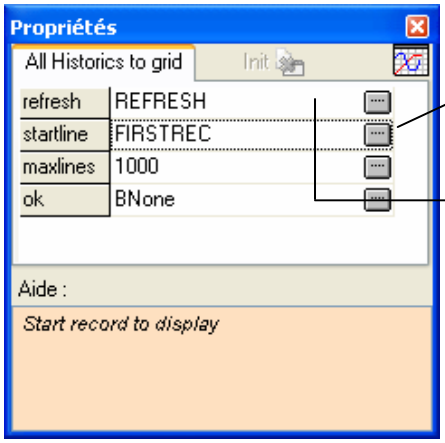
Ces composants sont livrés avec leurs sources, permettant d'effectuer des modifications ou des personnalisations.

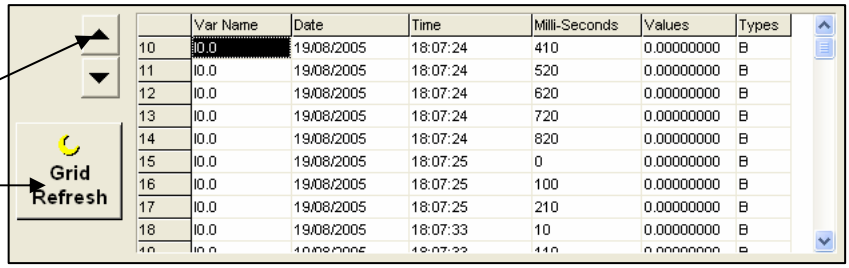
De gauche à droite :



AllHistoToGrid: Affichage du fichier historique dans une grille.
ExportHistorics: Exportation d'historique vers Excel.
VarHistoToGrid: Affichage d'un historique précis dans une grille.
TrendXY : Affiche d'un tableau de points.
PurgeHistorics : Purge du fichier historique à une date précise.
ShowEvents : Affichages des évènements (voir chapitre évènements).
ClearHistorics Remet à zéro le fichier des historiques (Histo.dbf).
StoreVars Sauvegarde dans un fichier une série de variables consécuti
LoadVars Charge d'un fichier une série de variables consécutives.
HistoFiFo Limite la taille du fichier Histo.dbf en créant un fifo sur disque
FifoValues Crée un FiFo pour les variables globales.

AllHistoToGrid





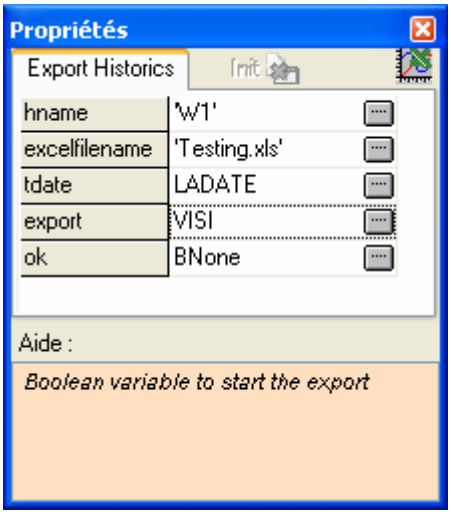
Le composant ALLHistoGrid permet de visualiser dans une grille l'ensemble des enregistrements du fichier Histo.dbf. Ici le composant UP/Down en haut à gauche permet de faire varier le paramètre StartLine permettant ainsi d'avancer la visu dans la grille.
Ce composant n'effectue pas de sélection sur les historiques affichés.

Refresh : Variable de commande de rafraîchissement de la grille par le contenu du fichier Histo.dbf.
Ok : Résultat du rafraîchissement (à remettre à false par programme).
StartLine : Premier enregistrement à afficher dans la grille.
MaxLines : Nombre d'enregistrements à afficher dans la grille successivement à startline.

☞ **Note** : En faisant varier *Startline*, l'utilisateur peut faire avancer ou reculer une zone de visualisation dans le fichier des historiques (Histo.dbf).

ExportHistorics

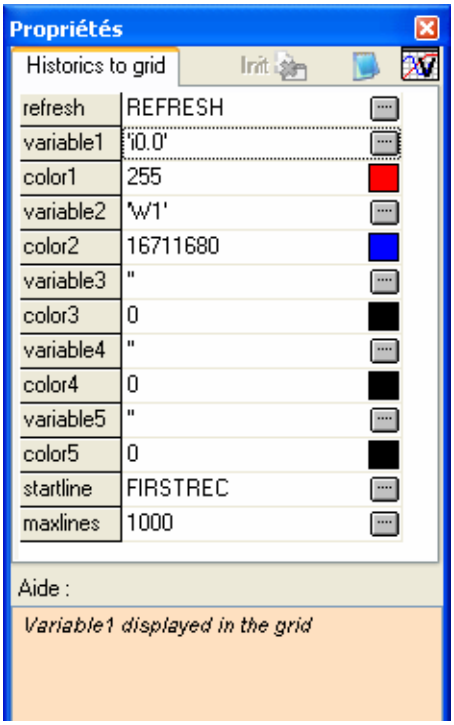
Le composant ExportHistorics permet d'exporter dans un fichier texte au format Excel l'historique d'une variable jusqu'à une date spécifiée.

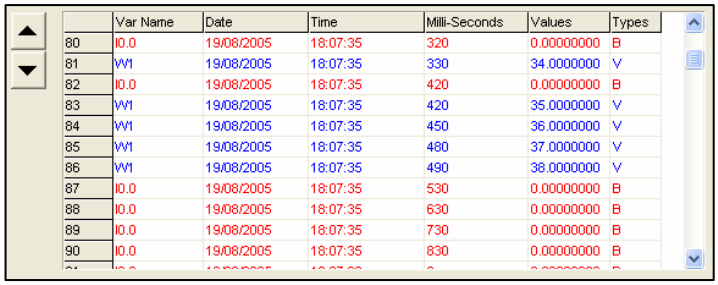


HName : Nom de la variable historisée à exporter.
ExcelFileName : Nom du fichier Excel à créer.
TDate : Date limite d'export de la variable. Tous les enregistrements inférieurs ou égaux à cette date seront exportés dans le fichier Excel. SI TDate='' alors le système prends automatiquement la date du jour.
Export : Variable de commande d'export.
Ok : Résultat de l'export. Le composant renvoi True si au moins un export à été effectué.

☞ **Note** : Pour le choix de la date, vous pouvez utiliser le composant ChoiceDate dans l'onglet Alarms.

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

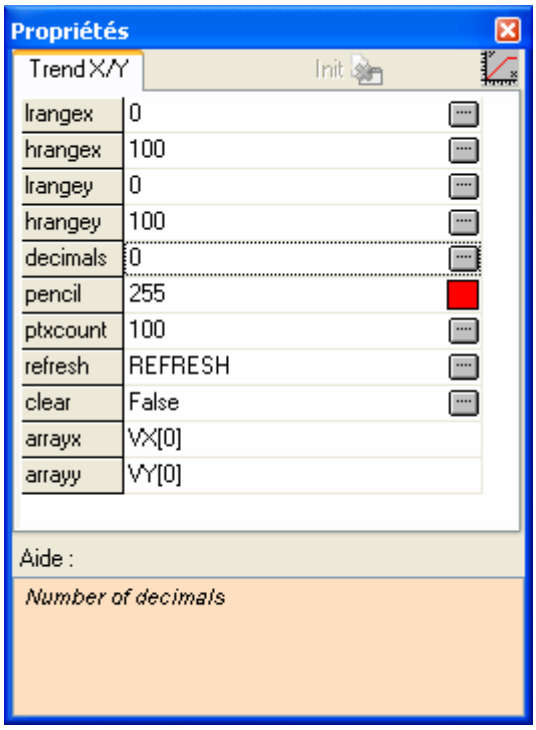
Refresh : Variable de commande de rafraîchissement de la grille par le contenu du fichier Histo.dbf.

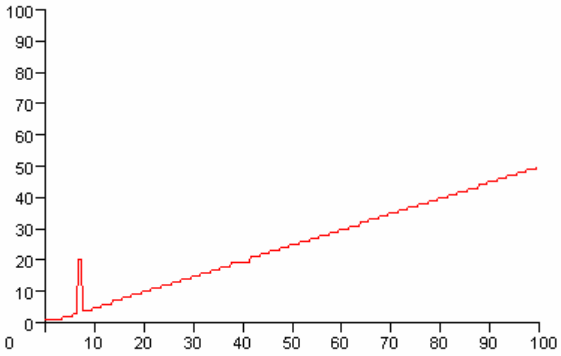
VariableX : Nom des 5 variables historisées à visualiser dans la grille.
ColorX : Couleur du texte de la variable trouvée.
StartLine : Premier enregistrement à partir duquel il faut chercher dans l'historisation des variables (Fichier Histo.dbf).
MaxLines : Nombre d'enregistrements à scruter dans le fichier Historique (Histo.dbf) et ceci à partir de startline.

☞ **Note** : En faisant varier *Startline*, l'utilisateur peut faire avancer ou reculer une zone de recherche dans le fichier des historiques (Histo.dbf).

TrendXY

Le composant TrendXY permet d'afficher un tableau de points Y par rapport à un tableau de points X sous la forme d'une courbe. If faut au préalable initialiser les deux tableaux de points X et Y et préciser au composant le nombre de points à afficher.

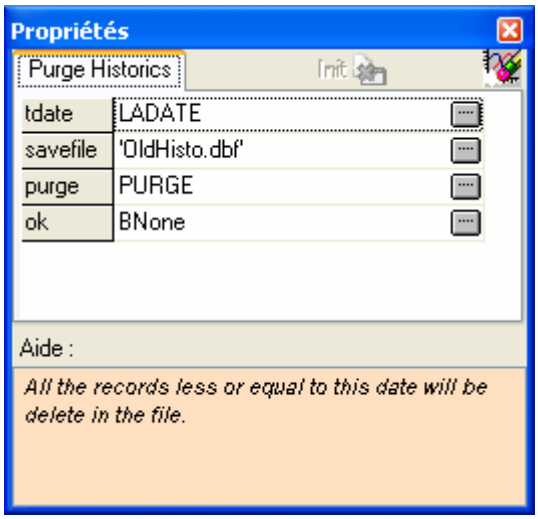




LRangeX : Valeur minimale de l'échelle en X.
HRangeX : Valeur maximale de l'échelle en X.
LRangeY : Valeur minimale de l'échelle en Y.
HRangeY : Valeur maximale de l'échelle en Y.
Decimals : Nombre de décimales affichées sur les graduations des axes X et Y.
Pencil : Couleur du tracé.
PtxCount : Nombre de points dans le tableau des valeurs X.
Refresh : Rafraîchissement du tracé.
Clear : Effacement du tracé.
ArrayX : Tableau des valeurs en X. (Type Array[0..X] of Real).
ArrayY : Tableau des valeurs en Y. (Type Array[0..X] of Real).

Purge historics

Le composant PurgeHistorics permet de purger le fichier Histo.dbf jusqu'à une date spécifiée. Les enregistrements effacés sont enregistrés dans un fichier de sauvegarde de type Dbase (extension .DBF).



TDate : Date butée d'effacement des enregistrements. Tous les enregistrements **inférieurs** à cette date seront effacés du fichier.

SaveFile : Nom du fichier de sauvegarde des enregistrements effacés.

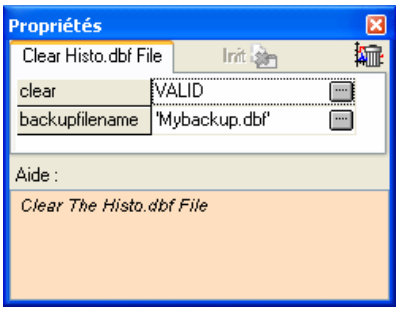
Purge : Variable de commande de purge.

Ok : Résultat de la purge.
Le composant retourne True si des effacements ont été effectués.

La fonction PurgeHisto utilisée par le composant est disponible dans la Vpu **ObjHisto**.
Pour créer le fichier de sauvegarde dans un répertoire différent de celui de l'application, il faut préciser le chemin complet du fichier.

ClearHistorics

Le composant ClearHistorics permet de copier le fichier Histo.dbf courant dans le paramètre BackupFilename si celui-ci n'est pas chaîne vide puis de remettre à zéro le fichier Histo.dbf courant.

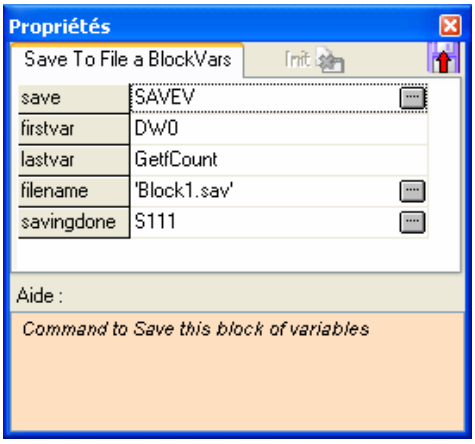


Clear : Permet de sauvegarder le fichier Histo.dbf courant ceci si *Backupfilename* spécifie un nom de fichier, puis d'effacer tous les enregistrements dans le fichier histo.dbf.

BackupFilename : Nom du fichier de sauvegarde des enregistrements effacés.

StoreVars

Ce composant permet de sauvegarder rapidement dans un fichier une zone mémoire de variables globales débutant à FirstVar et finissant à la variable déclarée avant LastVar.



Exemple dans les variables Globales.

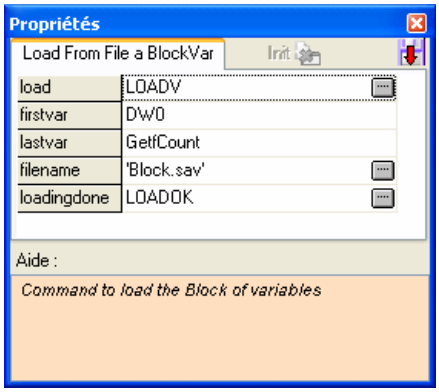
```
DW0 :  Integer; //*** Première variable
DW1 :  Integer;
DW2 :  Integer; //** Dernière variable
GetfCount :  Boolean; //** talon non
sauvegardé
```

Save permet de déclencher la sauvegarde.
FirstVar : Indique la première variable à sauvegarder.
Lastvar : Indique la dernière variable +1 à sauvegarder. Cette variable sert de talon mais n'est pas sauvegardée.

FileName Nom du fichier de sauvegarde des variables.
SavingDone : Indique que le fichier de sauvegarde à été créé.

LoadVars

Ce composant permet de charger rapidement depuis un fichier une zone mémoire de variables globales débutant à FirstVar et finissant à la variables déclarée avant LastVar (Inverse du composant Storevars)



Exemple dans les variables Globales.

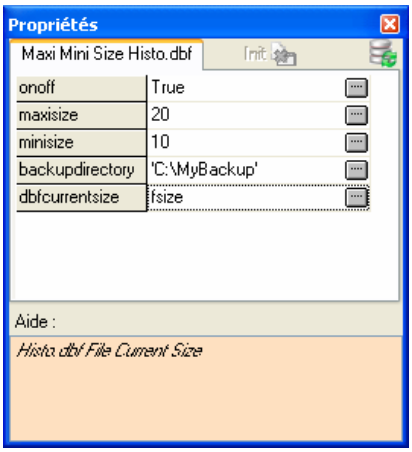
```
DW0 :  Integer; //*** Première variable
DW1 :  Integer;
DW2 :  Integer; //** Dernière variable
GetfCount :  Boolean; //** talon non sauvegardé
```

Load permet de déclencher le chargement.
FirstVar : Indique la première variable à charger.
Lastvar : Indique la dernière variable +1 à charger. Cette variable sert de talon mais n'est pas chargée.

FileName Nom du fichier source.
SavingDone : Indique que le fichier à été chargé.

HistoFiFo

Ce composant permet de limiter la taille du fichier courant des historiques (Histo.dbf). La taille variera d'un minimum jusqu'à un maximum. Les 2 valeurs étant un multiple de 100KBytes.



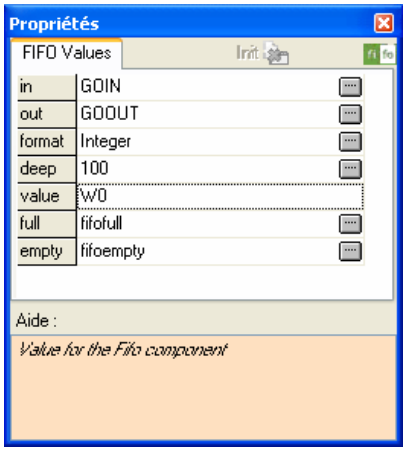
OnOff : Permet de mettre en service ou non le contrôle du Fifo sur le fichier Hist.dbf courant.
MaxiSize Taille maximum du fichier Histo.dbf exprimé en MégaBytes.
MiniSize Taille minimum du fichier Histo.dbf exprimé en MégaBytes.
BackupDirectory : Indique si la chaîne n'est pas vide un repertoire ou sera stocké les valeurs purgées. Le nom du fichier de purge sera la date courante avec extension .DBF (format DBase). Le format DBase peut être lu par Visual I/O ainsi que les logiciels de bureautique tel que Excel.

DbfCurrentSize : Optionnel indique la taille courante du fichier Histo.dbf

Note :
Backupdirectory peut être une constante comme ci-dessus ou une variable contenant le chemin du répertoire ou les valeurs du fichier historique purgé seront stockés sous un nom formaté comme ceci :
`Date.DBF` ex : pour la date 12/03/2006 le nom du fichier sera `12_03_2006.DBF`.
Si il est nécessaire de purger plusieurs fois à la même date, alors le système créera `12_03_2006[2].DBF` `puis 12_03_2006[3].DBF` et ainsi de suite.
Si **Backupdirectory** est une chaîne vide, alors aucune sauvegarde des valeurs purgées ne sera faite.

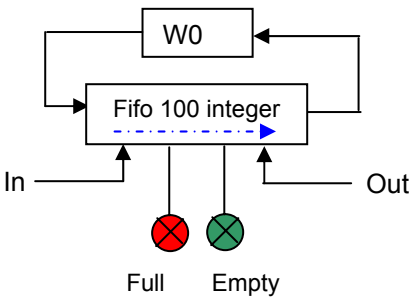
FiFoValues

Ce composant permet de créer une pile FIFO (First in First Out) Premier entré est premier sorti.
Cette pile permet alors de stocker des valeurs pour ensuite les restituer.



In : Commande pour entrer une nouvelle valeur dans le FiFo.
Out : Commande pour sortir une valeur du FiFo. Cette valeur est la plus ancienne qui est entrée.
Format : Format de la variable.
Deep : Nombre de valeurs dans le FiFo.

Full : Vrai si le fifo est plein.
Empty : True si le fifo est vide.



Programmation sur les fichiers historiques

Connaître la taille courante du fichier des historiques

Dans la VPU OBJHISTO la variable **FILEHISTOSIZE** indique la taille en octets (Byte) du fichier histo.dbf
Pour utiliser cette variable dans un programme, n'oubliez pas de déclarer **uses ObjHisto**;

Purger par programme le fichier des historiques

```
Uses sysplc,Windows,OBjHisto,Microbase;
// *** Value2 ************************************
SObject Procedure Value2;
Var
  R : real;
Begin
 SetValue(FileHistoSize);          // *** Affiche la taille
 If FileHistoSize>10000000 Then    // *** Taille fichier histo >10 Méga
 Begin
   R:=Encodetime(12,00,00,0);   // ** Codage pour 12 Heures
   R:=1;                        // ** Codage pour 24 Heures
   R:=Now - ((R)* 120);         // ** Codage pour 120 jours en moins
   PurgeHisto ( Datetimetostr(R),'Backup.dbf'); // Purge le fichier Historique
 End;
End;
```

Arrêter par programme la tache d'historisation

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

Pour le redémarrage : `RunTask('PCHISTO',True);  //* Run Realtime task PCHISTO`

Exporter vers excel

La fonction **ExportHisto** contenue dans la librairie **OBJHISTO.VPU** permet d'exporter un historique d'une variable à une date précise ou à partir d'une date précise et jusqu'à la fin du fichier historique.

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

Le paramètre DataExclusive de type booléen peut être passé ou non. Visual I/O affectera automatiquement un état False à ce paramètre manquant (paramètres optionnels).

Exemple 1 exportation de l'historique de la variable 'W0' à partir de la date spécifiée dans la chaîne de caractères *LaDate*. Toutes les historisations de W0 à partir de *Ladate* et supérieure seront exportées dans le fichier FExcel.XLS.

```
Var
  LaDate : String;
If button Then ExportHisto('W0','FExcel.XLS',LaDate); // Visual I/O affecte false
                                                      au parameter manquant.
```

Exemple 2 exportation de l'historique de la variable 'W0' à la date spécifiée dans la chaîne de caractères *LaDate*. Toutes les historisations de W0 à *Ladate* et seulement seront exportées dans le fichier FExcel.XLS.

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

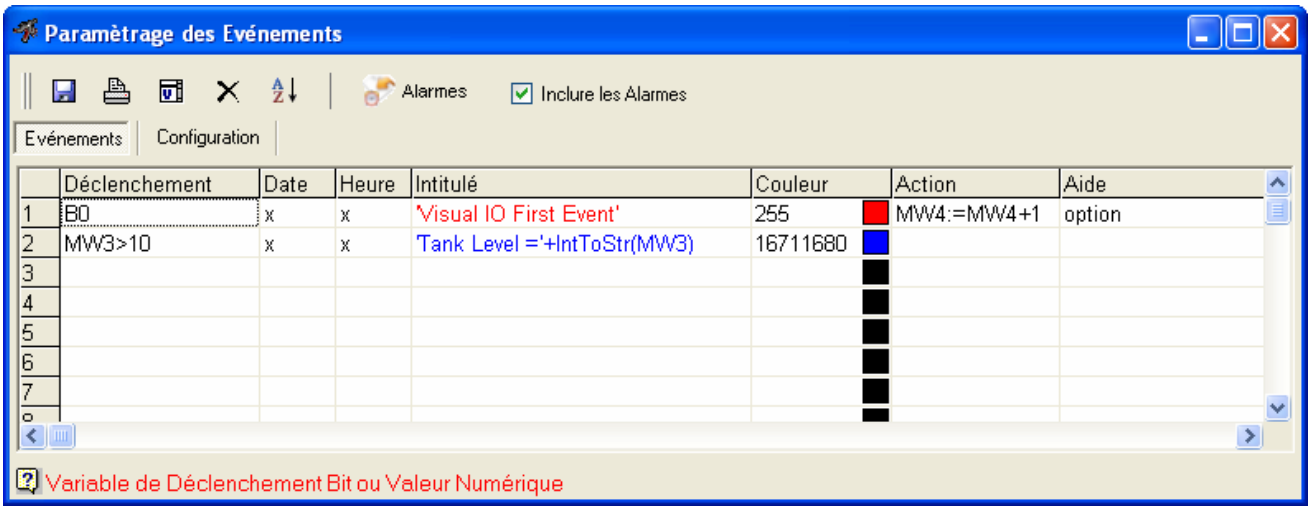
Appel d'Excel par programme

Vous pouvez faire afficher un fichier DBF directement dans Excel. **Shellexecute** lance Excel affiche (open) imprime (Print) le fichier spécifié. Ici Histo.dbf le fichier des historiques.

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

Evènements

Une grille de paramétrage apparaît permettant de programmer des événements . Les événements sont assimilables aux alarmes. Cependant un évènement peu générer une action simple ou complexe. Les évènements comme les alarmes sont visualisables en run dans un composant spécial (listbox) . Le résultat de ce paramétrage est contenu dans le fichier PCEVENT.pas et PCEVENT.VPU (compilé).

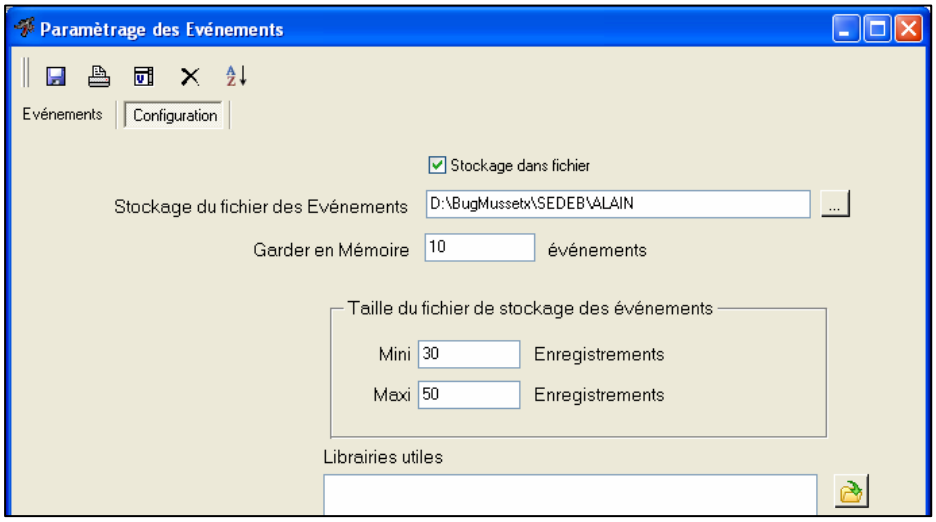


Bouton Alarmes : Permet d'ajouter les alarmes à la grille de paramètrages des évènements. Cette fonctionnalité est interessante si vous devez modifier les textes, couleurs des alarmes ...
Attention si par la suite vous ajoutez des alarmes dans le gestionnaire d'alarmes, cette grille ne sera pas mise à jour automatiquement. Il faudra ajouter manuellement le ou les textes des alarmes supplémentaires.

Case à cocher Inclure les Alarmes : En cochant cette case, les alarmes courante seront ajouter automatiquement lorsque vous sauvegarderez la grille. Les alarmes n'apparaitrons pas dans la grille de paramètrage des évènements.

Déclenchement : Simple état binaire à tester (sur front montant) ou expression complexe. Le résultat de cette expression déclenche l'évènement.
Date, Heure : Si X alors la date ou l'heure est inséré avant l'intitulé.
Intitulé : 1 / chaîne de caractères simple **entourée de guillemets**
2 / concaténation de chaînes de caractères
Couleur : Couleur d'affichage du texte dans la liste
Action ; Optionnel. Petit programme effectué quand l'évènement arrive. Ce programme est effectué seulement au front montant du déclenchement.
Aide : texte d'aide associé (non traité dans le composant de visualisation).

Onglet Configuration

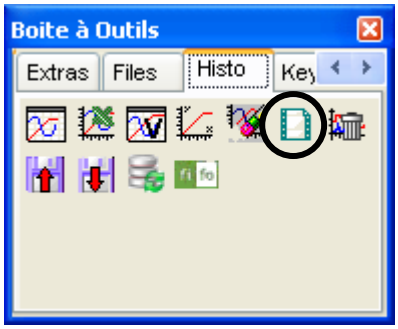


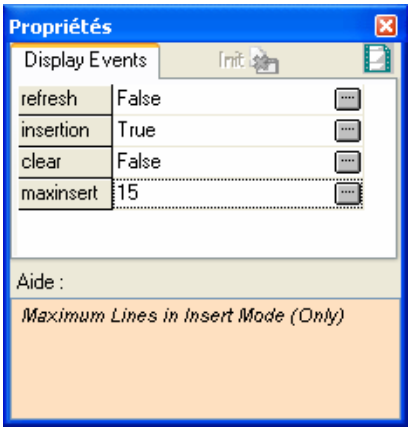
Stockage dans fichier : Si cette case est cochée les évènements sont sauvegardés dans le fichier EVENT.TXT. Sinon aucun stockage sur disque n'est effectué malgré les paramétrages suivants.
Stockage du fichier des événements : Si aucun répertoire n'est spécifié, le répertoire courant de l'appli est utilisé.Les événements sont stockés dans un fichier texte nommé 'EVENT.TXT directement exploitable sous excel.
Garder en mémoire : permet de définir le buffer mémoire des X derniers événements.
Taille du fichier de stockage (EVENT.TXT) détermine le nombre maxi et mini d'événements à conserver dans le fichier.
Librairies utiles : permet d'utiliser des procédures et fonctions externes dans les colonnes intitulés et action.

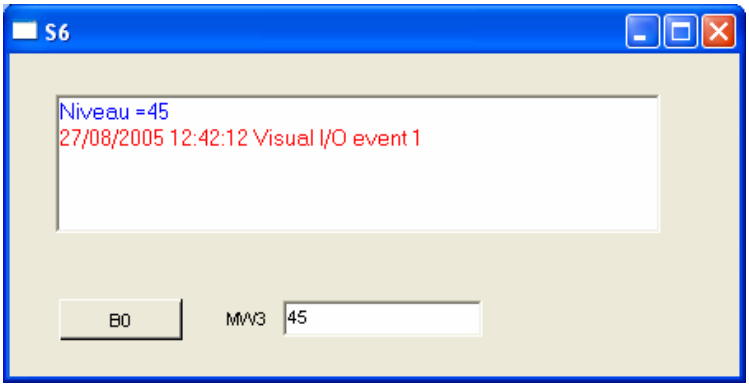
☞ **Note** : Afin de mieux comprendre les actions générées, le programme source généré PCEVENT.PAS est consultable.

Composants pour la gestion des événements

Display events







Refresh : Permet de rafraîchir la liste des événement seulement lorsque insertion est à false.

Insertion : Défini le mode d'affichage.

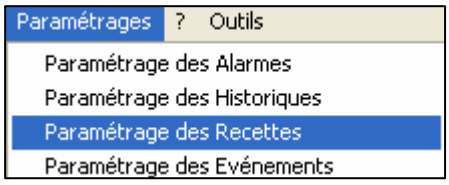
True : Seul les événements au fil de l'eau apparaissent à concurrence de Maxinsert.
Si Maxinsert=-1 alors la liste se remplie indéfiniment.
False : Les événements au fil de l'eau apparaissent mais tous les événements passés sont aussi listés. Les X événements à garder en mémoire défini dans l'onglet configuration sont toujours affichés.

Clear : Efface la liste.

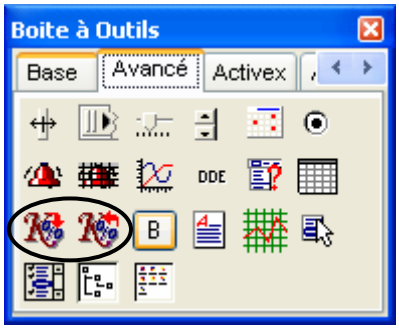
Maxinsert : Maximum de lignes visibles seulement valide lorsque insertion vaux true

Les évènements se comportent comme un historique.
C'est-à-dire que si un évènement disparaît le texte reste affiché dans la liste.

Gestion de recettes simples



Dans le menu principal de l'éditeur cliquez sur Paramétrage des Recettes



Paramétrage des recettes

Les recettes permettent de définir la liste des variables qui doivent êtres affectées par le chargement d'une **formule**.
Le paramétrage d'une recette n'est juste qu'une déclaration des variables à utiliser par les formules.

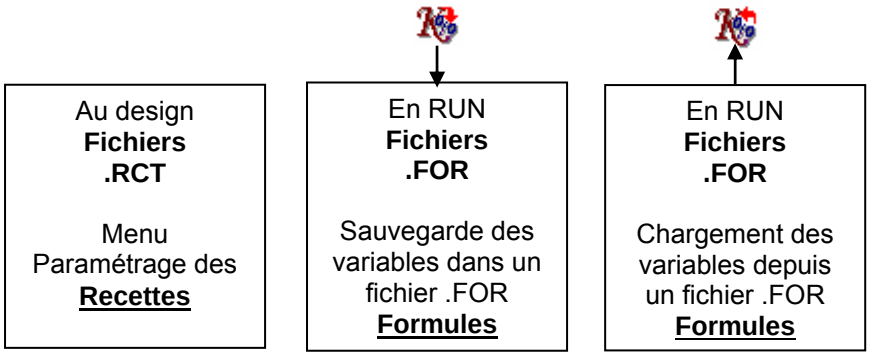
Les recettes contiennent des ingrédients définis : Ici dans la recette Rct1 sont définis les ingrédients à utiliser.

Action de type booléen
Ladate de type Numérique
Maxi de type Numérique
Mini de type Numérique.

Le composant sauvegarde d'une formule permettra de sauvegarder dans un fichier de nom donné les valeurs de ces 4 ingrédients.

Le composant chargement d'une formule permettra de télécharger les variables de ces 4 ingrédients avec les valeurs sauvegardées auparavant par le composant sauvegarde formule.

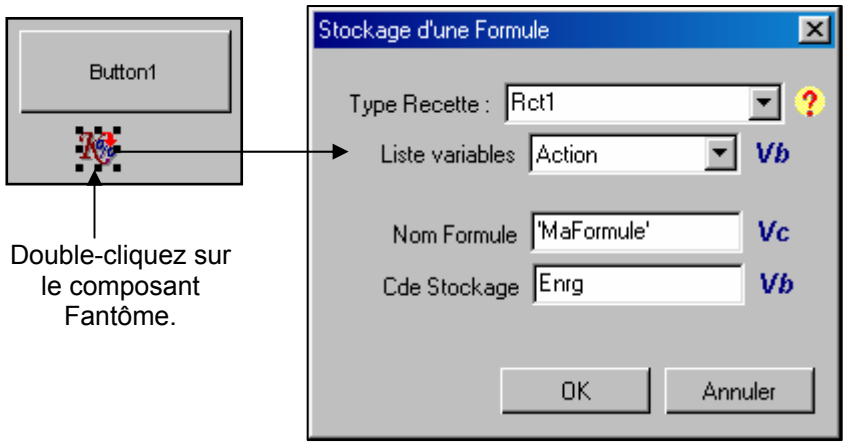
La structure



Enregistrement d'une Formule

Cet expert permet d'enregistrer des valeurs, Bit ou chaînes de caractères dans un fichier.

Pour enregistrer une formule, celle-ci doit connaître les ingrédients quelle doit enregistrer. Ceux-ci sont décrits dans une recette qui à été constituée par le programmeur à l'aide de la boite de paramétrages *'Paramétrages des recettes'* (Rubrique Paramétrages dans la barre de commande principale de l'éditeur de Visual I/O).



Choisissez le **Type de recette** sur laquelle le format de la formule doit s'appuyer.

Nom Formule : Entrez le nom du fichier sous lequel la formule doit être enregistré.
Si ce nom est déjà connu ajouter un Guillemet avant et après (constante chaîne de caractères).
Si ce nom ne sera déterminé qu'en RUN indiquez alors le nom de la variable chaîne qui contiendra le nom final du fichier de sauvegarde.

Cde Stockage : Indiquez une variable booléenne déclenchant la sauvegarde de la formule (enregistrement sur disque) sur front montant.

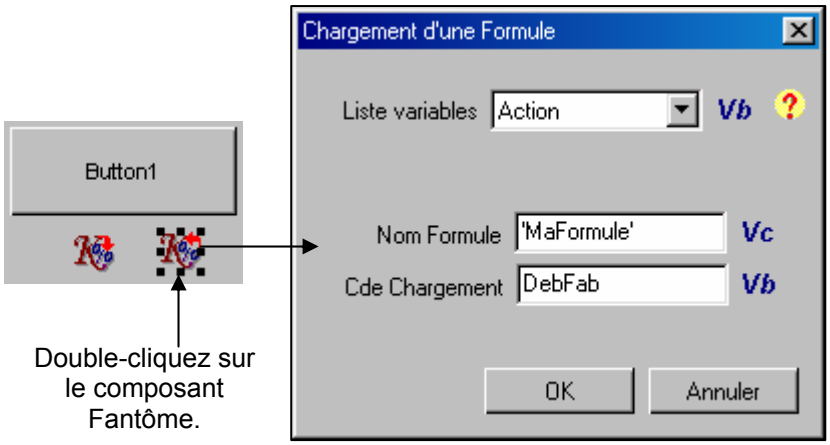
Détails :
Par exemple si la recette contient les variables Action, LaDate, Maxi et Mini, alors cette commande enregistrera dans le fichier formule les valeurs de ces variables à l'instant de la commande de stockage.

En général on ne connaît jamais à l'avance les valeurs pour constituer une recette.
Il faut les déterminer en **Run** puis les enregistrer pour les restituer au moment du lancement d'une fabrication par exemple.

☞ **Note** : l'avantage de l'enregistrement et du chargement de formules réside dans le fait que ces traitements s'effectuent en tache de fond, et que le synoptique contenant ces objets ne doit pas forcément être visibles.
Le déclenchement se faisant par des bits, il est possible d'associer un Grafcet pour l'enchaînement des chargement de formules.

Chargement d'une Formule

Cet expert permet de télécharger les ingrédients (variables) d'une recette depuis un fichier formule donnée (fichier stocké sur disque).

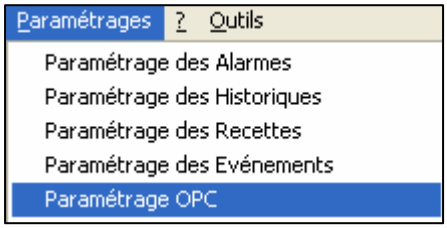


Le nom de la formule est spécifié dans le champ **Nom Formule**.

La commande de chargement des variables constituant la recette est effectué quand la variable booléenne spécifiée dans **Cde Chargement** est à True (Front Montant).

☞ **Note** : Nous n'avons pas besoin de spécifier le nom de la recette sur laquelle s'appuie cette formule car le nom de la recette est enregistrer dans le fichier formule lors de sa constitution.
Le Fichier .FOR stocké sur disque connaît les variables de l'application à télécharger avec ses valeurs.

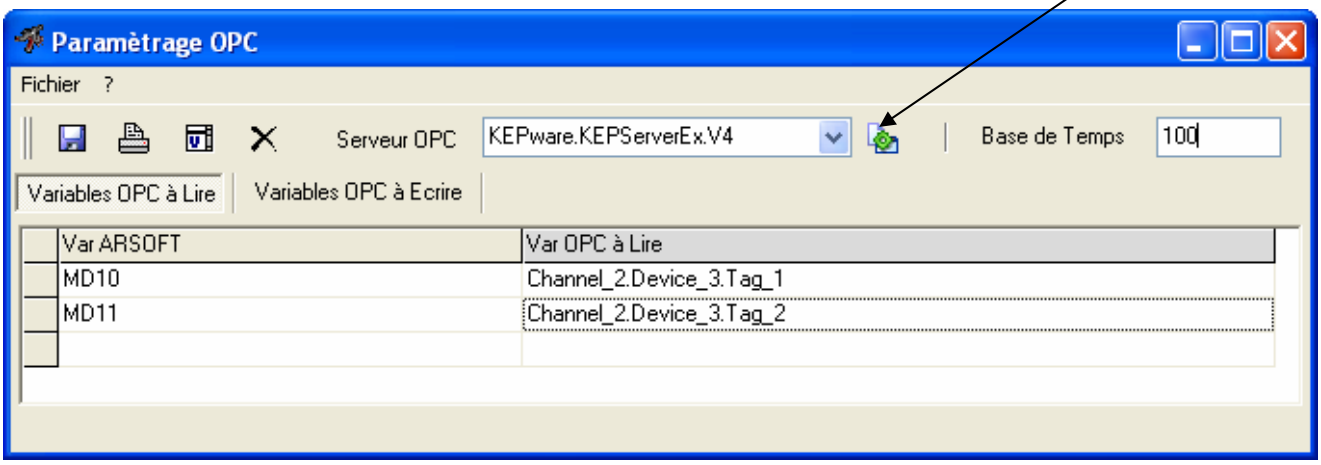
Gestion des liaisons OPC client



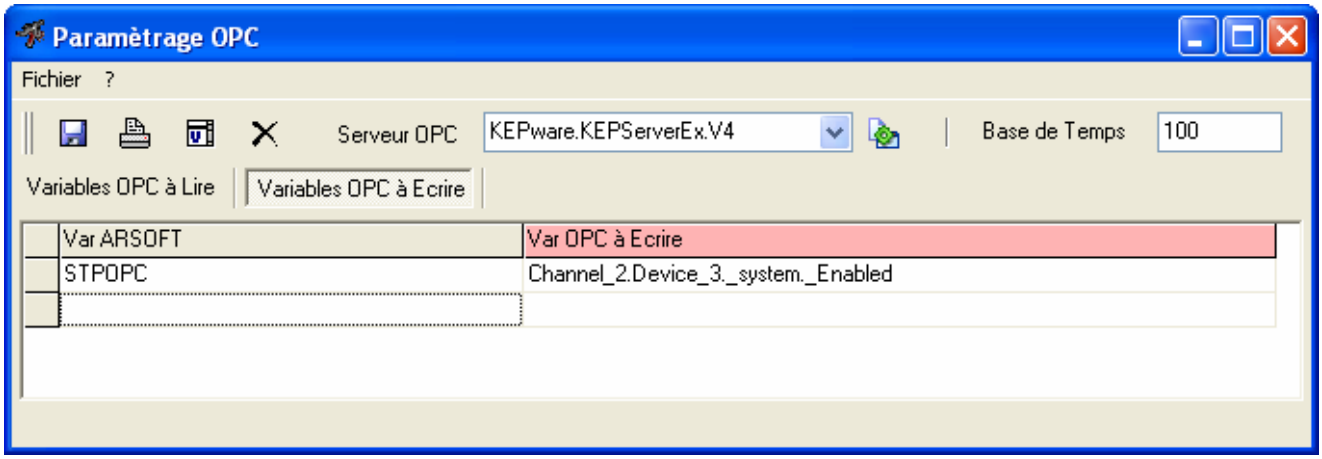
Une grille de paramétrage apparaît permettant de programmer les liaisons OPC. L'application ainsi générée sera une application OPC **client**. **OPC** (OLE for Process Control) est un standard de communication basé sur la technologie OLE/COM (comme d'habitude !). OPC définit des objets standards, des méthodes et des propriétés bâtis sur le concept COM (Component Object Model) pour permettre à des serveurs de données "temps réel" comme les DCS, les API et les équipements de terrain de communiquer leurs données vers des applications clientes OPC.

Les serveurs OPC les plus utilisés sont bien évidement les drivers de communication qui permettent d'échanger des données entre les automates programmables et les applications de supervision.

Après installation d'un serveur OPC, celui-ci est enregistré dans la base de registre de Windows. Afin de connaître son nom et de s'assurer de sa présence, cette boite de paramétrage permet de **scanner tous les serveurs OPC présents** dans votre PC.



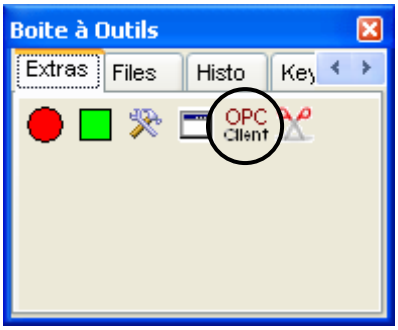
La connexion OPC permet de lire et d'écrire des variables dans le serveur OPC. Un onglet de lecture est disponible mettant en correspondance des variables Visual I/O et des variables du serveur.
Base de temps : Permet de définir à fréquence doit se faire la remontée des données du serveur vers l'application Visual I/O. Si cette valeur est égale à 0 la remontée des valeurs se fait à 10ms.
Var ARSOFT : Variable Visual I/O recevant l'état de la variable OPC.
Var OPC à Lire : Nom de la variable OPC à lire dans le serveur. Ce nom est défini dans le serveur OPC que vous allez utilisé (pas de règles établies).

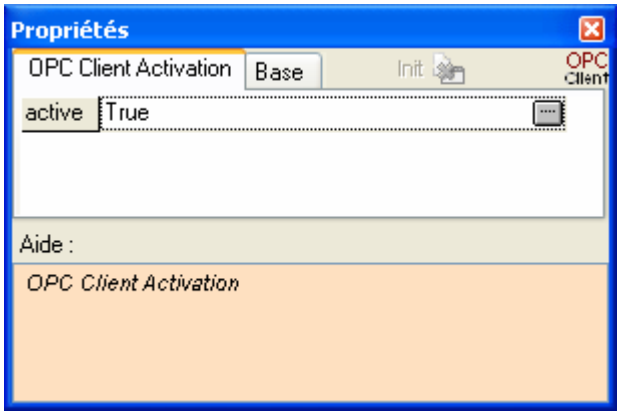


La partie écriture est similaire à la lecture (2 colonnes). Après sauvegarde un fichier source est généré et compilé automatiquement de **OPCCLIENT.PAS et OPCCLIENT.VPU**. Le fichier source est consultable, modifiable et recompilable à souhait. Le programme OPCCLIENT.VPU est ajouté automatiquement à la liste des taches prioritaires de l'application (voir menu Utilitaires/Constitution application).

Activation de la connexion OPC client

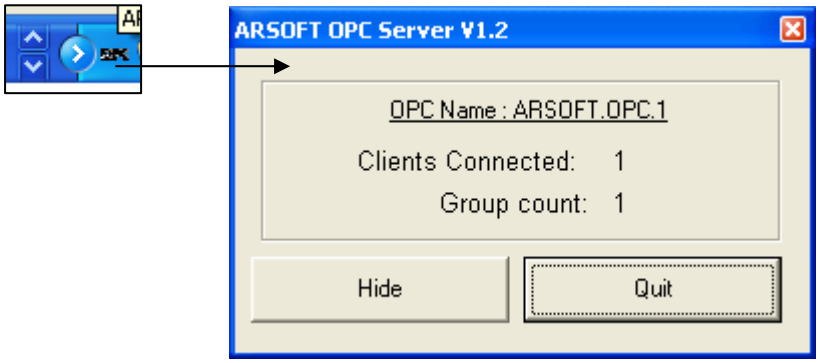
Pour que la lecture et l'écriture des variables OPC du serveur soit effectuée, vous devez placer le composant OPC Client sur le synoptique principal afin que celui-ci soit exécuté cycliquement. Ce composant permet aussi à l'aide de sa propriété de stopper ou mettre en service le lien OPC.





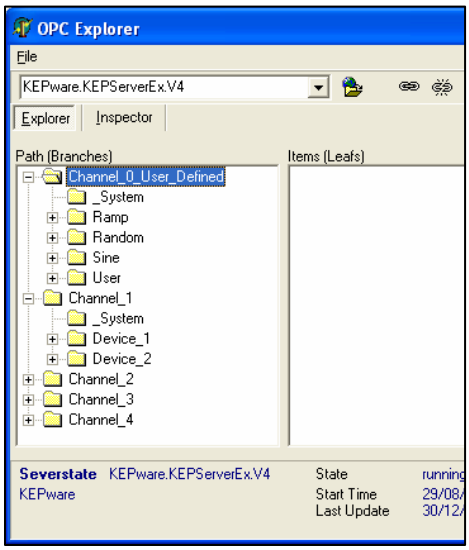
Rendre votre application serveur OPC

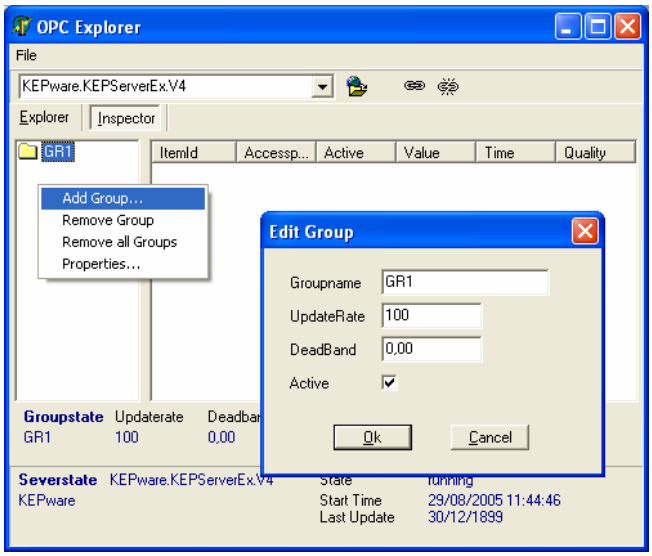
Afin de transformer votre application en un serveur OPC il est nécessaire de télécharger le serveur OPC sur le site ARSOFT (www.arsoft-int.com *rubrique resource, Serveur OPC pour applications Visual I/O & Visual PLC*). Le serveur OPC pour Visual I/O est un simple exécutable indépendant qui se place de la barre de tache de windows et qui donne l'interface OPC à toutes vos variable Globales.
Le nom du serveur est ARSOFT.OPC.1 et les variables sont les noms des variables globales déclarées dans votre application (W0, MD10, B0, BP1 etc.).



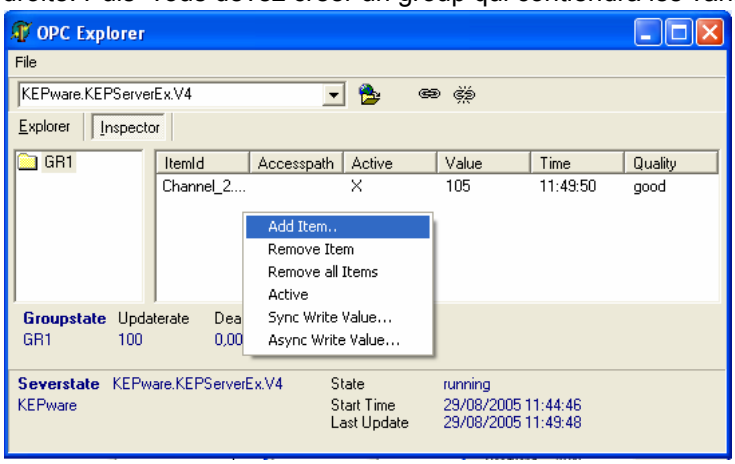
ARSOPC.EXE est un simple exécutable qui transformera votre application en un serveur OPC.
Ce serveur sera lancé automatiquement lorsqu'une application cliente demandera une lecture ou écriture de variable globale.
Si le serveur est lancé, celui-ci est visualisé par une petite icône dans la barre de taches de windows.

Un utilitaire (Client OPC) est livré avec le pack OPC serveur. Cet utilitaire **dOPCExplorer.EXE** permet de se connecter à n'importe quel serveur et de visualiser les variables échangeables, les lire ou les écrites.





Cet utilitaire vous permettra de connaître la liste des variables disponibles. Avant de pouvoir lire ou écrire dans un serveur il faut le choisir dans le Combobox supérieur, ce connecter par un des 2 boutons en haut à droite. Puis vous devez créer un group qui contiendra les variables à lire ou écrire (norme OPC).



◄ Click droit sur la grille de droite pour ajouter une variable à lire ou écrire.
Ce menu flottant permettra aussi de forcer l'état d'une variable.

☞ **Note** : Cet utilitaire étant gratuit, celui-ci est livré tel quel et à la responsabilité de l'utilisateur.

Le Générateur de rapport

Visual I/O offre la possibilité d'imprimer des rapports.
Ce générateur de rapports à des fonctionnalités modestes mais très souvent suffisantes pour les applications.
Ces rapports peuvent contenir des :

- Lignes
- Rectangles
- Rectangles à coins arrondis
- Cercles
- Ellipses
- Textes
- Valeurs
- Bitmaps ou Métafiles
- Supports (Frames)
- Courbes

Pour créer un format de rapport, **Un synoptique sert de support**.
Dans ce synoptique servant de cadre de rapport, vous pouvez placer des éléments graphiques Statiques ou Dynamiques. Ce synoptique devra faire partie de votre application finale (voir chapitre Constitution Application). Donc l'inclure dans la liste des synoptiques.

Eléments statiques :
Ce sont des figures graphiques de décors tel que des lignes, un bitmap logo, un support, un fond plan etc..

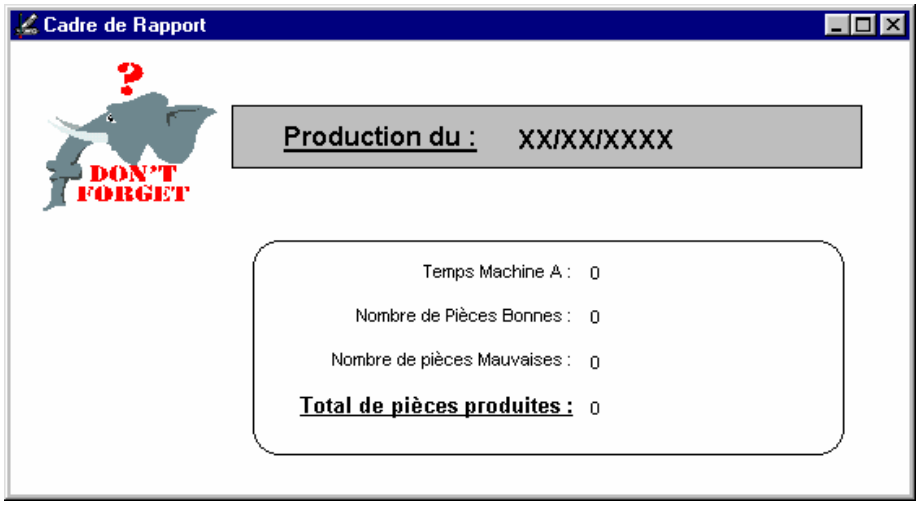
Eléments dynamiques :
Ce sont des éléments tel que des textes ou valeurs qui auront une valeur inconnue au moment de la constitution du rapport. Ces éléments dynamiques recevront leurs valeurs juste avant l'impression du rapport ils sont paramétrés avec les variables respectives.

Tous les éléments sont actifs une fois avant impression. Ceci permet de cacher, afficher ou déplacer par paramétrage classique. Surtout d'imprimer votre rapport selon vos critères.

La résolution de l'imprimante sur laquelle vous allez imprimer est nettement supérieure à celle du synoptique servant à la constitution de votre rapport. Si le synoptique est trop petit pour placer tous les éléments, faites glisser un tracé graphique hors des limites du synoptique pour faire apparaître les barres de défilement de celui-ci. Placer les éléments statiques de votre rapport, puis faite des essais d'impression pour déterminer les limites d'impression.

Exemple de design d'un rapport

Sur ce synoptique nous avons positionné des éléments statiques de décor :
Un fichier Métafile '*Don't Forget*', des rectangles et des intitulés (Textes).
Des éléments dynamiques :
Un texte XX/XX/XXXX qui devra recevoir la date avant impression.
4 valeurs numériques sans décimales qui sont affectées par les boites de paramétrage à des variables réelles.



Si par programme vous décidez d'imprimer ce rapport, les éléments dessinés ci-dessus apparaîtront tel quel sans modification sauf pour les valeurs numériques qui seront affectées par les valeurs en cours. Si le synoptique s'appelle MonRapport il doit être intégré dans votre application finale pour servir de cadre d'impression car votre exécutable doit le trouver (*voir constitution d'une application*).

La commande d'impression du rapport ne peut être fait que par programme.

Exemple de commande d'impression :

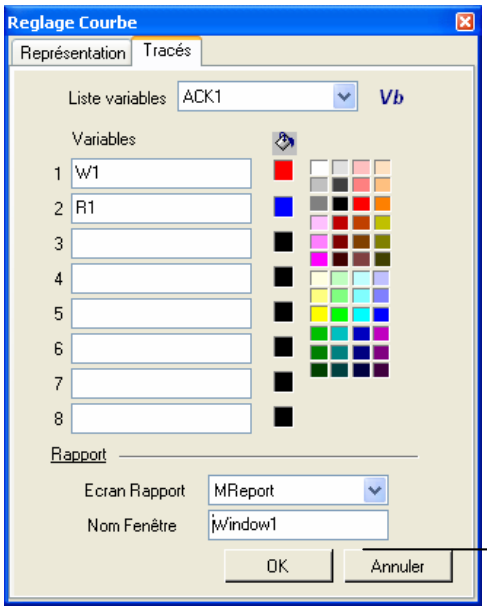
```
If Button Then  PrintReport('MonRapport') ;
If Envoie Then PrintReport('S8') ;
```

Le rapport est imprimé sur l'imprimante connectée par défaut sous Windows.

Vous pouvez effectuer une hardCopy d'écran par la fonction **HarCopy** ;
Pour passer votre imprimante en paysage utilisez la fonction
SETPRINTERFORMAT (Portrait : Boolean) ; Si Portrait=True alors en portrait sinon en paysage.

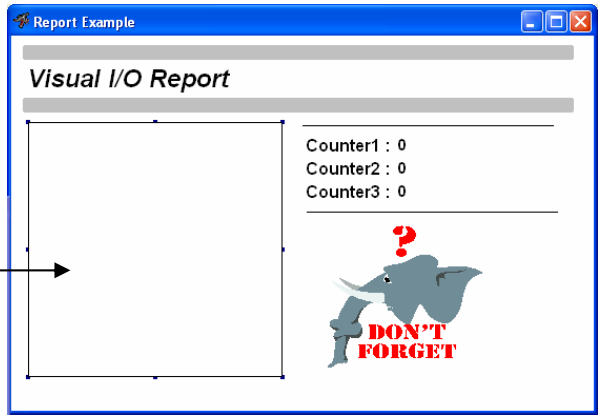
Intégration de courbes dans un rapport

Comme vu précédemment vous pouvez intégrer une courbe au fil du temps dans votre rapport (voir chapitre affichage simple au fil du temps) .



Le synoptique ci-dessous est tracé tout à fait normalement, mais ne sera pas affiché à l'écran il sert de cadre de rapport. Il sera imprimé et le cadre de fenêtre sera remplacé par les courbes présentes au moment de l'exécution de l'instruction **PrintReport**.
Le synoptique s'appelle MReport et la fenêtre Window1.

Le cadre ci-dessous est Window1 dans le synoptique MReport.

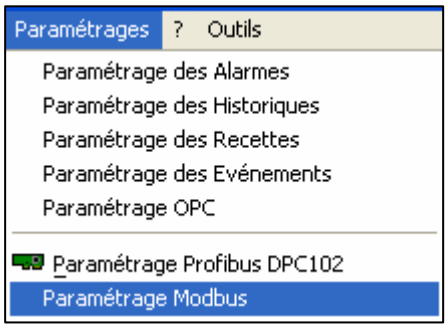


Incorporer une courbe par programme dans un rapport.

Si vous développez votre propre composant traceur de courbes, il est possible d'affecter cette courbe à un cadre window dans un rapport ceci par programme en utilisant la procédure **TrendToReport**

```
SObject Procedure Window1;
Var
   THDLE : Integer;
Begin
  If Start_Visu Then
  Begin
    THDLE:=CustomTrend(10,5,1,1,0,450); //** crée le cadre de courbe
    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); // affecte ce cadre au rapport Xreport
  End;
End;
```

Paramétrage Modbus



Cette grille de paramétrage permet de décrire les dialogues à effectuer cycliquement Ces dialogues peuvent lire/écrire des bits et des mots dans un esclave Modbus.
En ce qui concerne les mots, ceux-ci doivent êtres du type WORD (16 bits).
Avant d'appeler la grille de paramétrage MODBUS, vous devez déclarer une liste de Mots (WORD) et de bits recevant ou reflétant les variables à lire ou à écrire dans l'équipement esclave.
Après validation par le bouton OK les système génère PCMODBUS.VPU qui le driver final Modbus compilé.

☞ **Note** : *Un Click droit de la souris permet de copier/coller des lignes de paramétrage.*

Exemple : Je désire lire 10 mots dans un équipement esclave numéro 2.
1/ Déclaration de 10 Word en variables globales.

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

La première ligne permet la lecture dans l'esclave N° 2 de 10 mots (16 bits) consécutifs à partir de l'adresse 200 en Hexa. Les valeurs de ces mots seront rangées dans W1à W10. La colonne Variables permet de définir **le premier mot** de réception ou d'émission. Si plusieurs mots sont lus, le premier mot sera rangé dans celui spécifié puis les suivants dans les mots déclarés consécutivement dans les variables globales.

La colonne Commande permet de désigner ou non un bit qui sera déclencheur de trame Modbus.
Le Bit doit passer à True pour lancer la trame concernée (Lecture/Ecriture de bits ou de mots). Si le dialogue c'est correctement effectué, le système remet ce bit à false, permettant ainsi d'avoir un acquittement.

Le Driver MODBUS

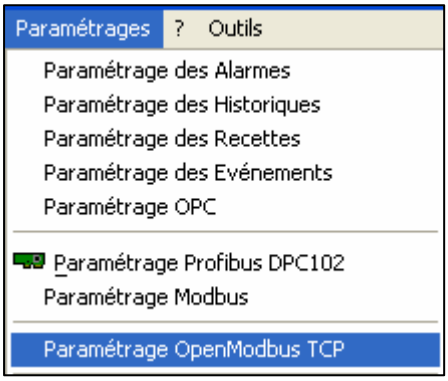
Après validation par le bouton Ok, Visual I/O génère un driver compilé tenant compte des paramétrages. Ce driver est ensuite implémenté automatiquement dans l'application en cours (Menu : constitution application / taches prioritaires). Ce driver sera exécuté en tache prioritaire. Assurant ainsi un dialogue à vitesse maximum.
Le nom du programme driver généré est PCMODBUS.VPU (le source PCMODBUS.PAS est visualisable).

Les commandes Modbus disponibles sont :
Lecture Mots, Lecture Mots interne, Ecriture Mot, Lecture Bit, Lecture Bit Interne et Ecriture Bit. Pour choisir le type de dialogue, double-cliquez sur le petit **bouton gris** à droite de la cellule.
Contrôler la communication est possible par le tableau de booléens
STATUSMODBUS : **ARRAY**[0..255] OF **Boolean**; déclaré dans les variables globales.
A chaque indice de tableau correspond une ligne de paramétrage effectuée. Si la ligne est correctement effectuée le booléen correspondant passe à True sinon false.

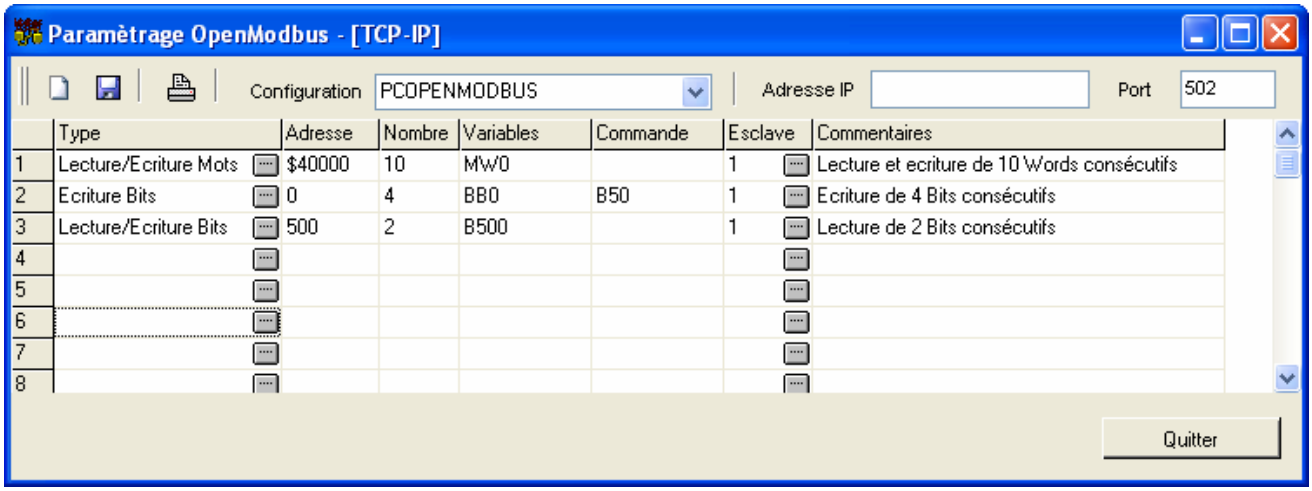
Annulation du driver

Allez dans le menu Utilitaires - Constitution de l'application et puis das l'onglet Taches prioritaires.
Double cliquez sur **PCMODBUS** afin d'éliminer le driver de la liste.

Paramétrage OpenModbus - Modbus sur Ethernet



Cette grille de paramétrage permet de décrire les dialogues à effectuer cycliquement Ces dialogues peuvent lire/écrire des bits et des mots dans un esclave Modbus. Cet esclave est connecté à la prise RJ45 de votre PC (prise réseau)
Avant d'appeler la grille de paramétrage MODBUS, vous devez déclarer une liste de Mots (WORD) et de bits recevant ou reflétant les variables à lire ou à écrire dans l'équipement esclave.
Après validation par le bouton OK le système génère PCOPENMODBUS.VPU qui le driver final Modbus compilé.
Le fichier de configuration qui apparaît dans la grille de paramétrage est PCOpenModbus.TCP.



☞ **Note** : *Un Click droit de la souris permet de copier/coller des lignes de paramétrage.*

Configuration : Nom du fichier de configuration visualisé dans la grille. Vous pouvez créer plusieurs fichiers de configuration correspondant aux différentes adresses IP. Le port est normalement fixé à 502 par la norme. Un programme VPU correspondant au nom de fichier visualisé est généré et compilé automatiquement. Il est inséré comme un nouveau driver dans votre application finale.

Adresse IP : Dans ce champ de saisie, entrez l'adresse ip de l'esclave openmodbus. Cette adresse est déterminée par un logiciel qui est fourni avec cet esclave.
Port : Généralement la valeur 502 selon la norme openmodbus. Visual I/O permet de changer de numéro de port si nécessaire.
La colonne **Esclave** permet de régler un numéro d'esclave qui n'est pas utilisé car l'adresse de l'esclave est précisée par l'adresse ip.

La première ligne permet de lire cycliquement 10 mots dans l'esclave à partir de l'adresse $40000 (40000 en hexa) et de ranger le résultat dans les mots Visual I/O commencant par MW0 et ceci consécutivement. Par contre si un de ces mots Visual IO change (par votre programme) une écriture vers l'esclave est généré.

La deuxième ligne permet d'écrire dans l'esclave 4 bits consécutifs à partir de l'adresse 0 avec les valeurs des bits BB0 et suivants. Cette écriture est effectuée lorsque le bit B50 est à True. Quand la commande d'écriture est effectuée par visual I/O le bit de commande est remis à False automatiquement par le système permettant ainsi d'effectuer un acquittement.

La troisième ligne permet la lecture cyclique de 2 bits commencant à l'adresse 500 dans l'automate. La valeur de ces 2 bits seront rangés dans le bit B500 et celui qui est déclaré après. Si Un des ces 2 bits change dans votre application alors une écriture (des 2 bits) est généré vers l'esclave.

La colonne Commande permet de désigner ou non un bit qui sera déclencheur de trame Modbus.
Le Bit doit passer à True pour lancer la trame concernée (Lecture/Ecriture de bits ou de mots). Si le dialogue c'est correctement effectué, le système remet ce bit à false, permettant ainsi d'avoir un acquittement.
Utilisez le composant ComparVars (Boite à outils onglet Times) pour détecter une zone de variables globales qui changerait d'état. Ce composant permet de passer un bit à true qui sera repris dans la grille de paramètrages.

Le Driver OPENMODBUS (PCOpenModbus.VPU)

Après validation par le bouton Sauver, Visual I/O génère un driver compilé tenant compte des paramétrages. Ce driver est ensuite implémenté automatiquement dans l'application en cours (Menu : constitution application / taches prioritaires). Ce driver sera exécuté en tache prioritaire. Assurant ainsi un dialogue à vitesse maximum. Le nom du programme driver généré est PCOPENMODBUS.VPU (le source PCOPENMODBUS.PAS est visualisable).

Les variables disponibles pour OpenModbus

Dès que vous ajoutez un traitement OpenModbus, le système ajoute automatiquement dans les variables Globales suivantes :

StatusOPENMODBUS : Array [0..400] Of Boolean; est un tableau de booléen (bits) indiquant le numéro de ligne correctement exécutée par le programme. En d'autres termes, à chaque indice du tableau correspond une ligne de paramétrage de la grille.
ComOPENMODBUSOK : Boolean; ce booléen indique si le socket windows de communication (Adresse ip et port) est ouvert. Ce booléen n'indique pas que l'esclave (E/S, Automate) OpenModbus est présent au bout du câble.

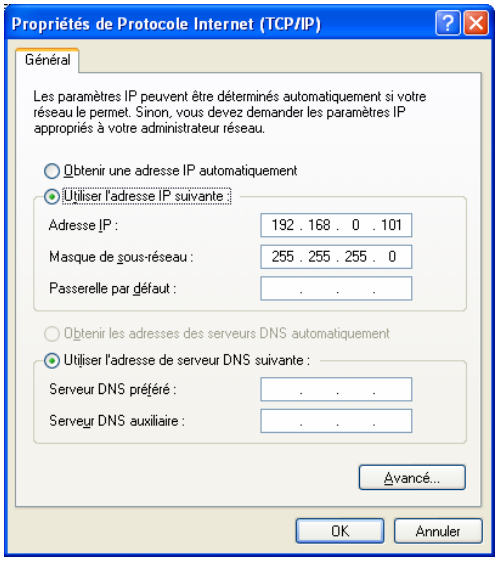
Note : Si la communication ne s'effectue plus (Cable débranché, esclave coupé), le réarmement et la reconnexion se remettent en service automatiquement.

Les variables utiles dans le driver (OpenModbus.vpu).

OpenModbusCounter : Integer; Ce mot indique le nombre de communications effectuées.
StopOpenModbus : Boolean=False; Ce booléen permet de stopper/reprendre par programme les communications avec l'esclave.

OpenModbusDisconnected : Boolean=False; Si ce booléen passé à True, cela indique que au moins une communication openmodbus à été effectuée et que l'esclave ne réponds plus (déconnection, manqué tension..).

Réglez l'adresse ip de votre PC (connection réseau)



Si l'esclave Openmodbus connecté à l'adresse ip suivante 192.168.0.100

Vous pouvez paramétrer la carte Ethernet de votre PC comme ci contre avec l'adresse 192.168.0.100 masque de sous-réseau 255.255.255.0

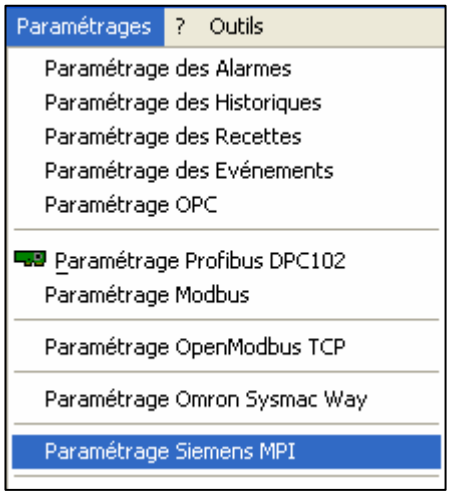
Le masque de sous réseau le plus courant, est celui que l'on utilise généralement 255.255.255.0

Ce masque de sous réseau va permettre aux ordinateurs ayant une adresse IP ayant 3 premiers octets identiques de communiquer ensemble.
Ex : l'ordinateur ayant l'IP 192.168.0.1 pourra communiquer avec l'autre ayant une IP telle que 192.168.0.2, **mais pas** 192.169.0.2

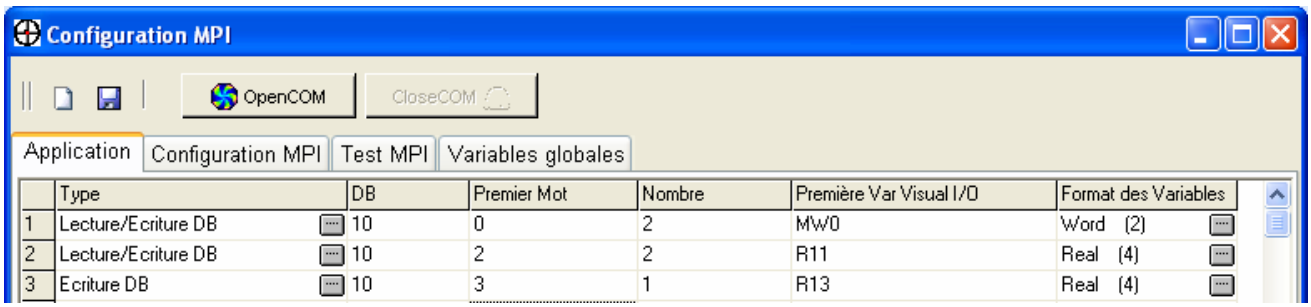
Types de communications possibles :

- Lecture Mots Internes : De l'esclave vers le PC
- Ecriture Mots : Du PC vers l'esclave
- Lecture Bits Internes : De l'esclave vers le PC
- Lecture/Ecriture Bits; : De l'esclave vers le PC et du PC vers l'esclave (si changements).
- Lecture/Ecriture Mots : De l'esclave vers le PC et du PC vers l'esclave (si changements).
- Ecriture 1 Bit : Du PC vers l'esclave
- Ecriture Bits : Du PC vers l'esclave
- Ecriture 1 Mot : Du PC vers l'esclave
- Lecture Statuts : De l'esclave vers le PC

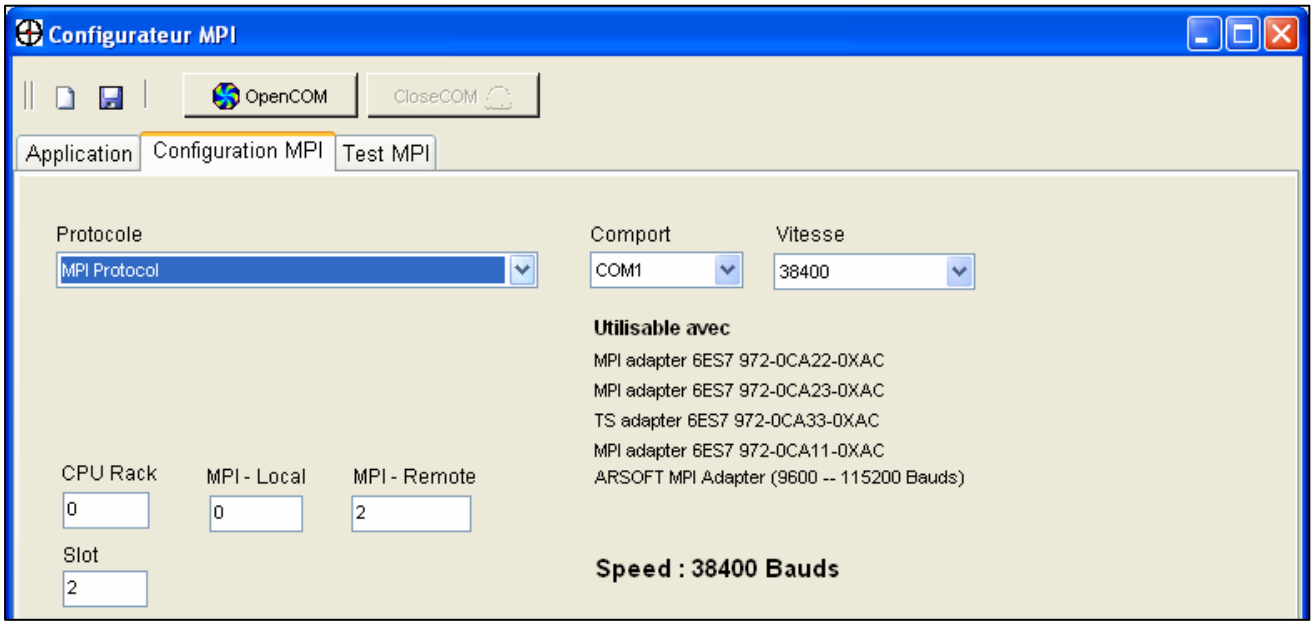
Paramétrage MPI – Série et Ethernet



Cette grille de paramétrage permet de décrire les dialogues à effectuer cycliquement. Ces dialogues peuvent lire et écrire des mots dans un esclave (automate) MPI. Cet automate est connecté à la prise RJ45 de votre PC (prise réseau) ou par la liaison série via un adaptateur MPI/Série
Avant d'appeler la grille de paramétrage MPI, vous devez déclarer une liste de Mots (Word, Integer ou real) recevant ou reflétant les variables à lire ou à écrire dans l'automate.
Après validation par le bouton Sauver le système génère **PCMPI.VPU** qui le driver final MPI compilé utilisable dans l'appli.
Le fichier de configuration qui apparaît dans la grille de paramétrages est PCMPI.CFG. Le configurateur est un programme externe à l'éditeur de Visual I/O. Ce configurateur à d'ailleurs été écrit en Visual I/O lui-même (sources disponibles sur demande).



Un premier onglet (Application) permet de paramétrer les lectures et les écritures à effectuer.
Les écritures se font seulement sur changement d'état, permettant ainsi d'optimiser la communication avec L'automate. Il est possible de lire ou/et écrire des DB, DI, Compteurs et Tempos. Pour les autres variables utilisez les procédures et les fonctions de la Vpu Mpiserial.vpu. Les formats des variables sont indiqués dans le combobox de la colonne *Format des Variables*. Il est donc possible dans un même DB d'utiliser des words, integers ou reéls. Il suffit alors de paramétrer plusieurs lignes.Les commandes **Lecture/Ecriture** permettent de lire cycliquement mais aussi d'écrire au même endroit dans l'automate. L'écriture se faisant seulement sur changement d'états des variables globales de votre application. Une correspondance de tables (Automate/Supervision) est ainsi directement possible.



La liaison série est utilisée avec les adaptateurs Siemens ou Vipa. Différentes vitesses sont possibles en fonction de l'adaptateur **série** utilisé. Généralement la vitesse appropriée est **38400 Bauds**.
Sinon la connection Ethernet de votre PC peut être utilisée soit directement avec la CPU ou à travers un adaptateur IBH ou hilscher.

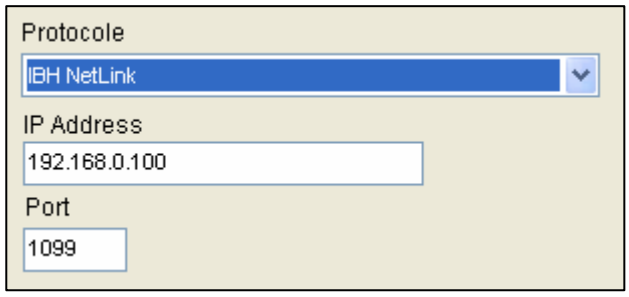
Les champs que vous devez obligatoirement remplir sont

CPU Rack : Numéro de rack de la CPU (coté automate).
MPI-LOCAL : Numéro MPI Coté PC (Généralement 0).
MPI-REMOTE : Numéro MPI Coté Automate.
SLOT : Numéro de slot dans l'automate.

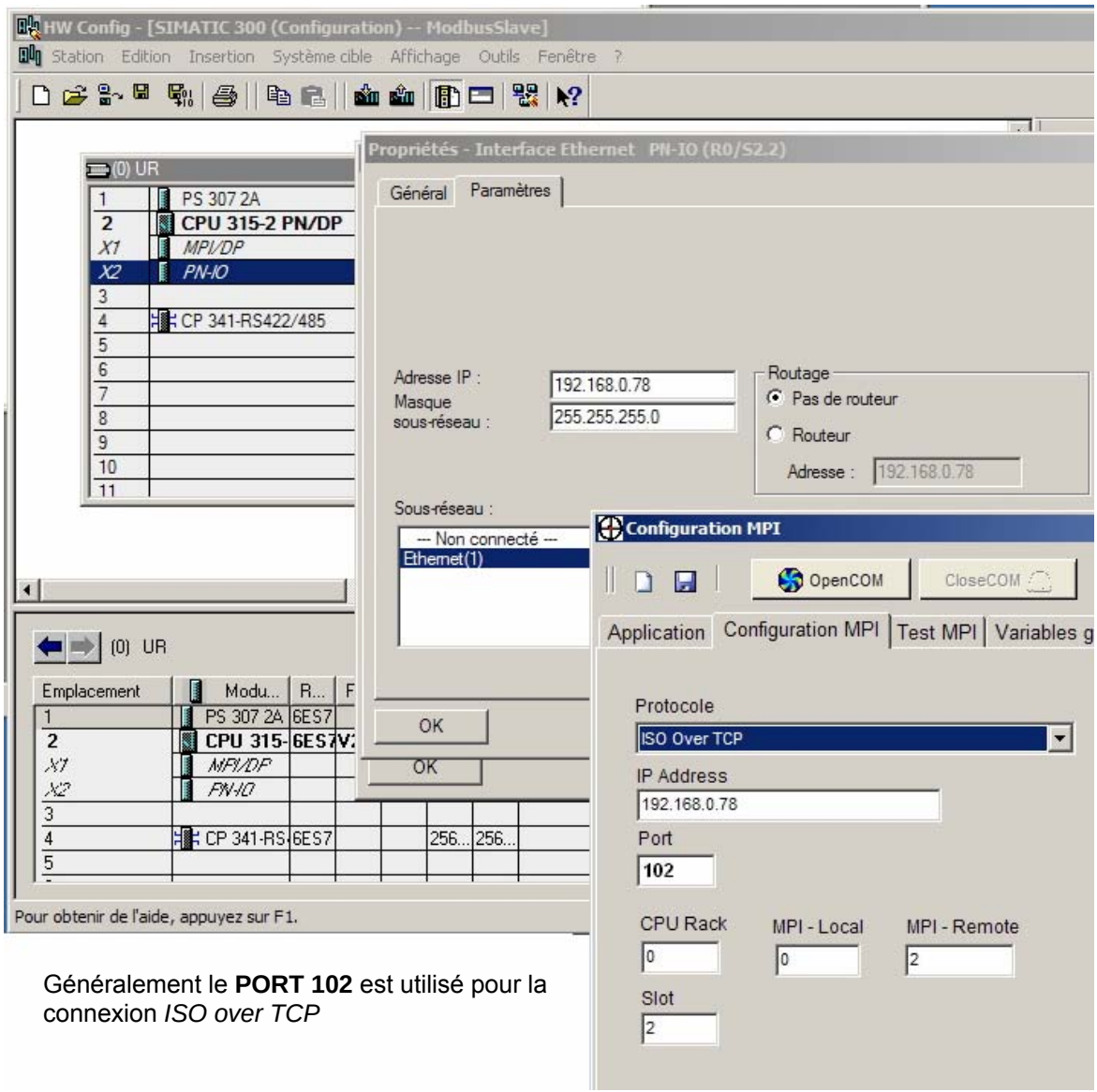
Concernant les réglages dans l'automate se référer à la doc siemens.

Si la connexion est effectuée en TCP/IP par un adaptateur ou directement relié à la cpu vous devez régler d'autres paramètres :

IP address : Est l'adresse IP sur laquelle est connecté votre adaptateur MPI ou la CPU automate.
Port : est le numéro de port utilisé dans votre PC Pour dialoguer en TCP.
Ces 2 paramètres supplémentaires, ne sont pas Spécifiques à la connexion MPI. Ils relèvent des Paramètres utilisés dans une connexion TCP/IP.

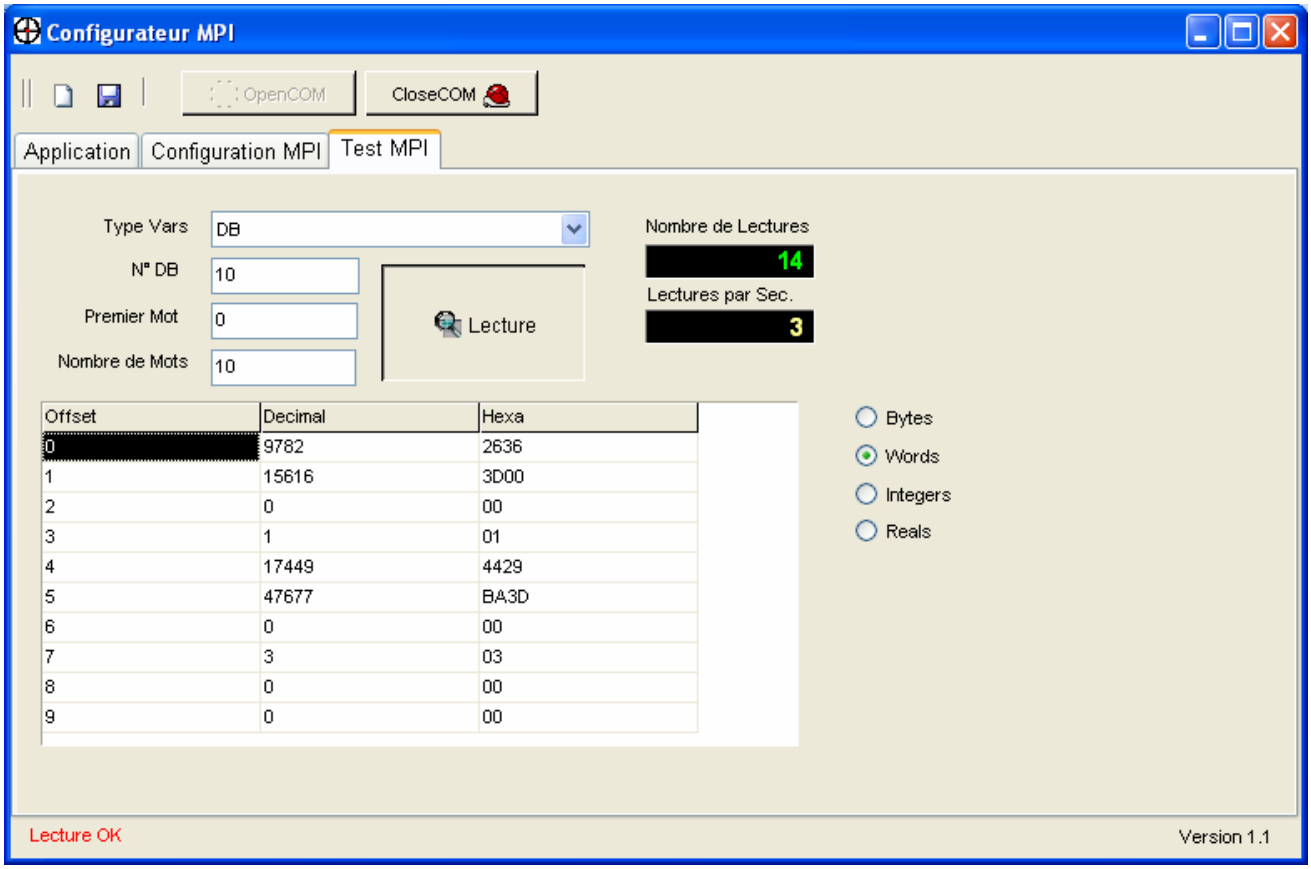


Communication avec Automate Siemens carte Ethernet intégrée type CPU 315 PN/DP



Généralement le **PORT 102** est utilisé pour la connexion *ISO over TCP*

Le dernier onglet test MPI, permet de tester la connexion directement dans le configurateur comme dans le configurateur Profibus. Le test tient compte du type d'adaptateur MPI paramètré (série, Ethernet).



1/ Cliquez sur le bouton OpenCOM pour établir la connexion avec l'adaptateur. Si la connexion est établie, ce bouton se grise automatiquement et autorise le bouton closeCom et lecture. Choisissez le type de variables à lire dans l'automate et son format ainsi que le nombre à lire. Les valeurs lues s'affichent dans la grille. Il est possible de modifier dans l'automate une variable, pour ce faire placez vous dans la colonne Décimal puis frappez une nouvelle valeur numérique dans la cellule choisie. Une trame d'écriture est envoyée automatiquement dans l'automate. Si la communication décroche (coupure automate, câble débranché..), cliquez sur le bouton CloseCom puis OpenCom pour réinitialiser l'ensemble de la communication avec l'automate.

Le driver PCMPI.VPU généré

Après sauvegarde et compilation correcte, le configurateur place automatiquement dans les taches prioritaires de votre application ce driver. Il ajoute aussi des variables supplémentaires dans les variables globales de votre application. Le fichier **MPIDLL.dll** est nécessaire au fonctionnement de votre application. Vous pouvez l'inclure dans votre executable (voir menu créer un exécutable puis dans onglet suppléments ajouter le fichier MPIDLL.dll. l'exécutable généré va extraire au lancement ce fichier et le copier dans C:\Windows pour utilisation)

COMMPIOK : `Boolean;` Indique si l'initialisation de la communication est effectuée :
1/ Ouverture du Port de communication série ou Ethernet
2/ Initialisation avec l'adaptateur MPI
3/ Connexion à l'automate.
Ce bit retombe à false si la connexion avec l'automate tombe en timeout. C'est-à-dire si Visual I/O envoie une trame et qu'il n'y a pas de réponse de l'automate.

MPILASTERROR : `String;` Cette variable renvoie un texte en anglais documentant l'erreur. Si aucune erreur n'est présente, cette chaîne est vide.

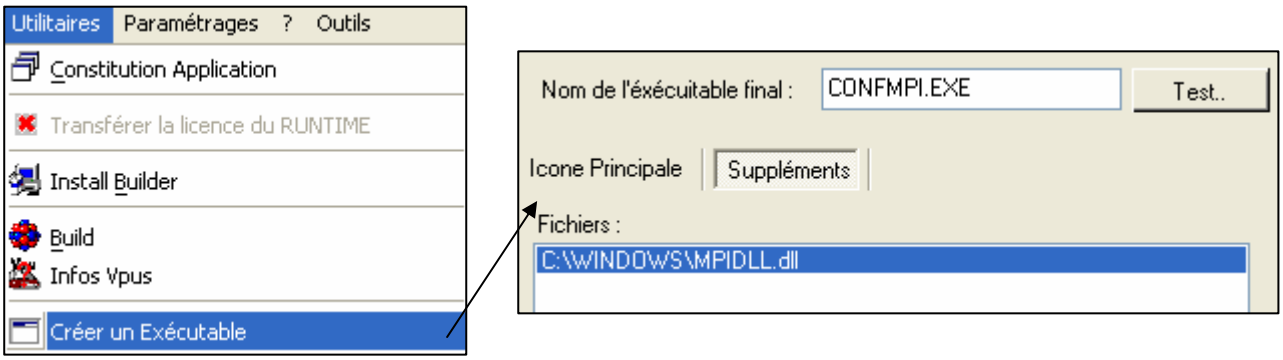
REARMMPI : `Boolean;` N'est plus utilisé, la déclaration est gardée pour compatibilité. Le réarmement MPI est maintenant automatique en cas de perte de celle-ci par time out ou câble débranché, automate coupé par exemple.

Procedure ReconnectMPI; est aussi disponible en interface de PCMPI.VPU.

Le source PCMPI généré est disponible dans la rubrique Programmes de l'éditeur de Visual I/O.

Important :
Pour effectuer une communication MPI, le PC à besoin de la DLL **MPIDLL.DLL** dans le répertoire C:\Windows. Normalement le configurateur MPI confMPI.EXE livré au lancement installe cette DLL.

Si vous livrez un exécutable à votre client, n'oubliez pas d'inclure **MPIDLL.DLL**

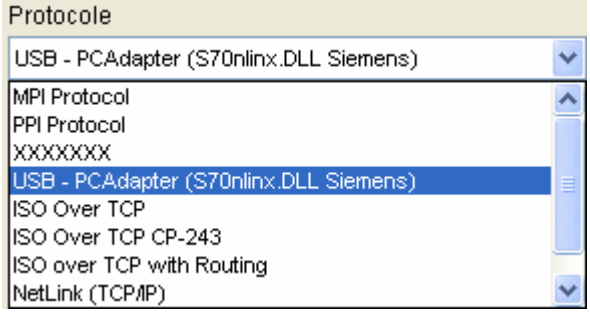


Le fait d'intégrer un fichier DLL dans votre exécutable final, permet au lancement de le renvoyer dans C:\Windows afin que l'application le trouve et l'utilise.

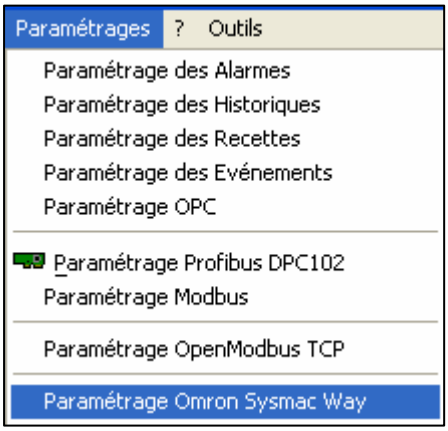
Le driver PCMPI.vpu utilise un thread indépendant de l'application, permettant ainsi de ne pas bloquer le reste de votre application en cas de time out avec l'automate. Dans tous les cas de figure, votre application restera fluide.

Les différentes connexions possibles :
PPI Protocol (série) automates Série 200.
MPI Protocol (Série)
ISO Over TCP (Ethernet)
IBH Netlink (Ethernet adapteur)
USB – PCadapter (MPI adapteru en usb de Siemens)

Sont 4 connexions les plus utilisées.



Paramétrage Omron



Cette grille de paramétrage permet de décrire les dialogues à effectuer cycliquement Ces dialogues peuvent lire/écrire des bits et des mots dans un automate Omron avec le protocole Sysmac Way.
En ce qui concerne les mots, ceux-ci doivent êtres du type WORD (16 bits).
Avant d'appeler la grille de paramétrage OMRON, vous devez déclarer une liste de Mots (WORD) et de bits recevant ou reflétant les variables à lire ou à écrire dans l'équipement esclave.
Après validation par le bouton OK les système génère PCOMRON.VPU qui le driver final SysmacWay compilé.

Les commandes Omron disponibles sont :

Lecture Mot(DM), Ecriture Mot(DM), Lecture Mot(IR) ,Ecriture Mot(IR), Lecture Mot(HR), Ecriture Mot(HR), Lecture Mot(TC) ,Lecture Bit Etat(TC).
Il est possible de lire jusqu'à 30 mots ou 120 bits consécutifs par ligne de commande dans la grille de paramétrage.

Les termes (DM), (IR), (HR) et (TC) définissent des zones mémoire dont la structure est détaillée dans le tableau suivant :

Zone de données		Mots	Bits	Fonction
IR	Zone d'entrée	IR000 à IR009 (10 canaux)	IR00000 à IR00915 (160 bits)	Ces bits peuvent être associés aux bornes d'E/S externes.
	Zone de sortie	IR010 à IR 019 (10 canaux)	IR01000 à IR 01915 (160 bits)	
	Zone de travail	IR 200 à IR 231 (32 canaux)	IR20000 à IR 23115 (512 bits)	Les bits de travail peuvent être librement attribués dans le prog.
SR	Drapeaux de fonctionnement	SR232 à SR255 (24 canaux)	SR23200 à SR25507 (384 bits)	Bits disponibles lorsque la fonction d' interruption ne sert pas
HR		HR00 à HR19 (20 canaux)	HR0000 à HR1915 (320 bits)	Bits sauvegardés en cas de coupure secteur
AR			AR0000 à AR1515 (256 bits)	Drapeaux de fonctionnement
LR	Liaison de données		LR0000 à LR1515 (256 bits)	Bits disponibles lorsque la fonction de liaison ne sert pas
Tempo /Compteur		TC000 à TC127 (Numéro des temporisations et compteurs)		On utilise les mêmes numéros pour les tempo. et compteurs
DM	Lecture/Ecriture	DM0000 à DM0999 DM1022 à DM1023 (1002 mots)		Mots sauvegardés en cas de coupure secteur
	Stockage des erreurs	DM1000 à DM1021 (22 mots)		Mots disponibles lorsque la fonction de stockage ne sert pas
	Lecture seule	DM6144 à DM6599 (456 mots)		Ré-écriture impossible à partir du programme
	Mise au point API	DM6600 à DM6655 (56 mots)		S'utilise pour stocker divers paramètres contrôlant le fonctionnement API.

Variables de diagnostique

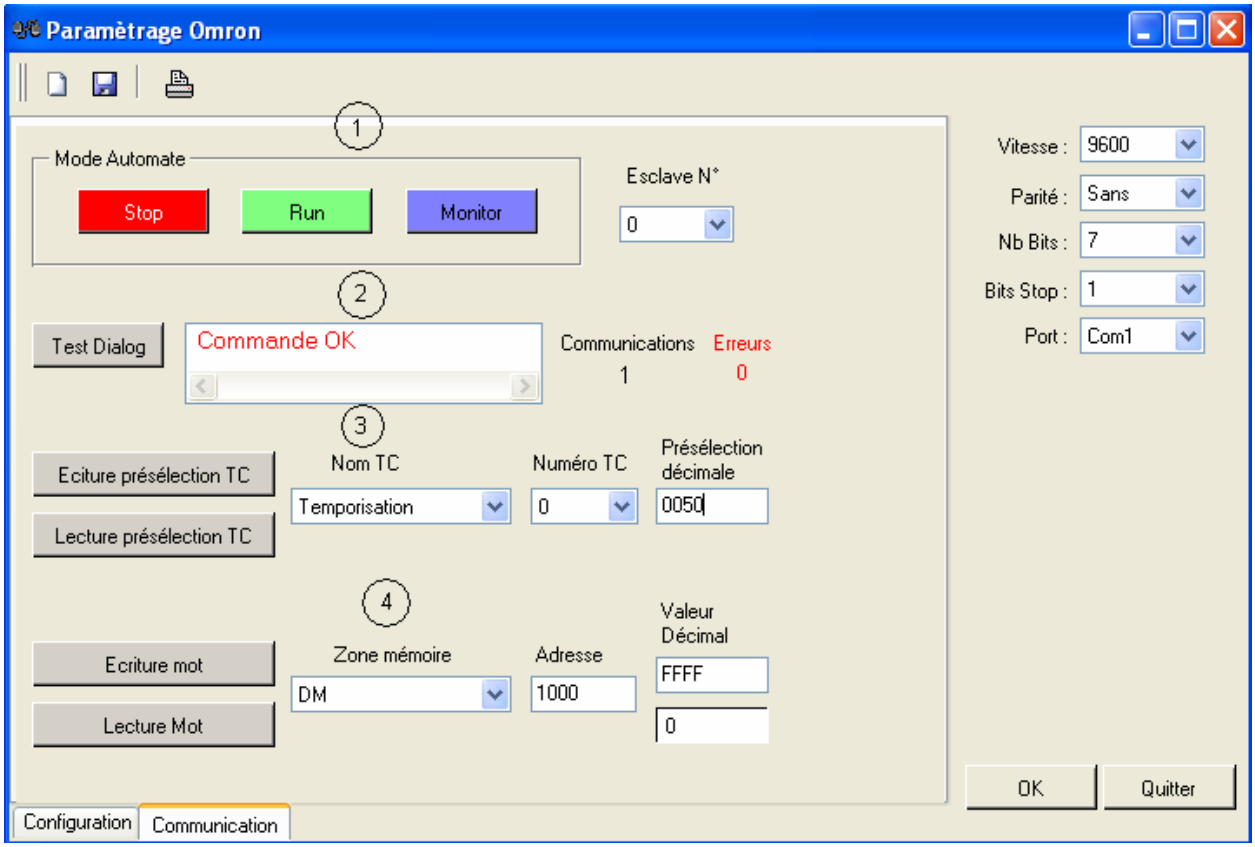
Après compilation des variables sont ajoutées automatiquement aux variables globales de votre application :

Le diagnostique des communications est disponible dans un tableau de booléens (StatusOmron) correspondant à la grille de paramétrage.
STATUSOMRON : ARRAY [0..255] of **Boolean**;
(Vrai si communication Ok ou communication en cours, Faux si erreur de communication).

COMOMRONOK : Boolean;
La variable ComOmronOk reçoit le résultat de la fonction Ouverture port. (Vrai si port ouvert, Faux si impossible d'ouvrir le port).

Un deuxième onglet propose une interface de dialogue direct depuis l'éditeur et avec un automate connecté sur la liaison série.

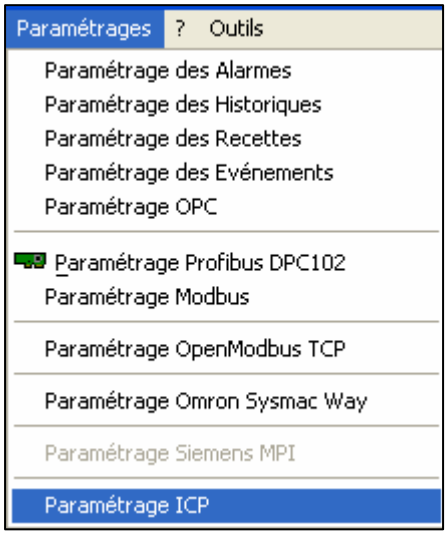
Test communication sur liaison série



Les fonctionnalités disponibles sont :

1 : Modification du mode automate. Le mode Monitor autorise certaines actions en Run.
2 : Test et diagnostique de communication avec l'automate.
3 : Lecture et écriture de présélection TC (Temporisation ou Compteur). La valeur à saisir et la valeur retournée est en décimal sur quatre digits.
4 : Lecture et écriture mot dans la mémoire zone IR, SR, HR, AR, LR, DM : La valeur à saisir et la valeur retournée est en décimal.
Lecture mot valeur courante zone TC (Temporisation ou Compteur) : La valeur à saisir et la valeur retournée est en décimal.
Lecture mot état zone TC (Temporisation ou Compteur) : La valeur à saisir et la valeur retournée est en binaire. Elle se lit de droite à gauche et représente l'état de sortie de quatre TC Consécutifs.

Paramétrage ICP



Cette grille de paramétrage permet de décrire les dialogues à effectuer cycliquement Ces dialogues peuvent lire/écrire des bits et des mots dans des modules ICP (et advantech compatibles)

Avant d'appeler la grille de paramétrage ICP vous devez déclarer une liste de Mots (WORD) recevant ou reflétant les variables à lire ou à écrire dans le ou les modules esclaves.
Après sauvegarde le système génère PCARSICP.VPU qui le driver final compilé. Celui utilise une DLL du contructeur qui est **I7000.dll** et **UART.dll**. Ces 2 librairies sont localisées dans le répertoire Windows ou Windows\System32, les répertoires *foure tout* de Windows.

Pour le test dans l'éditeur de VisualI/O, il est nécessaire d'avoir le DLL UART.DLL seulement.

Paramétrage des Modules ICP

Le paramétrage des modules ICP s'effectue à l'aide du programme 7000util.exe (Livré ou disponible sur site ARSOFT ou ICPDAS.COM).
Celui-ci vous permet de modifier l'adresse de l'équipement, sa vitesse de communication et éventuellement sa fonctionnalité (4-20 mA, +10 –10V etc pour les modules analogiques). Vous ne pouvez modifier les paramètres d'un seul module à la fois, la marche à suivre est la suivante :

- Couper l'alimentation des modules
- Ouvrir le module à paramètrer pour positionner le cavalier jaune sur le mode init
- Faire un shunt entre la borne GND et la borne INIT* du connecteur vert
- Rétablir l'alimentation des modules

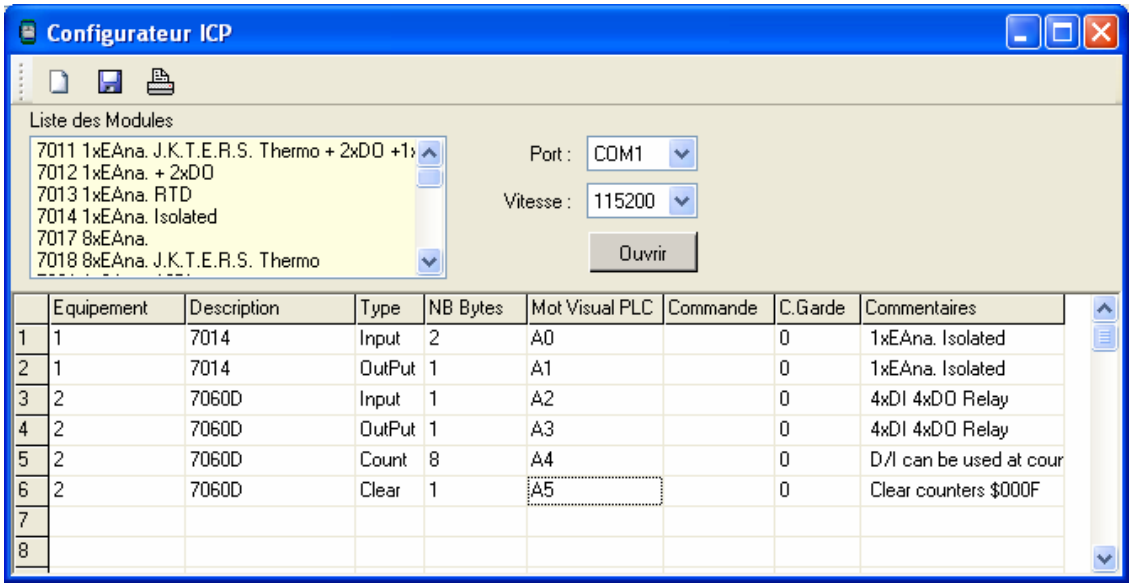
Lancer maintenant le programme 7000util.exe.

Configuration des modules

Amenez le type de module souhaité par double-clic dans la liste supérieure. Celui-ci se place en dernier affichant ainsi le nombre de bytes en Réception/Emission dont il a besoin (NB Bytes).

Placez alors le mot (ou premier Mot Visual PLC) dans la colonne correspondante.

Si plusieurs mots sont nécessaires, placez seulement le premier mot. Le reste des octêts seront rangés dans les mots déclarés successivement au premier mot notifié dans cette grille. Ceci en émission et en réception.



	Equipement	Description	Type	NB Bytes	Mot Visual PLC	Commande	C.Garde	Commentaires
1	1	7014	Input	2	A0		0	1xEAna. Isolated
2	1	7014	OutPut	1	A1		0	1xEAna. Isolated
3	2	7060D	Input	1	A2		0	4xDI 4xDO Relay
4	2	7060D	OutPut	1	A3		0	4xDI 4xDO Relay
5	2	7060D	Count	8	A4		0	D/I can be used at cour
6	2	7060D	Clear	1	A5		0	Clear counters $000F
7								
8								

ATTENTION :
Le type de variable **Integer** est impérativement utilisé pour les échanges avec un module digital .
Le type de variable **Real** est impérativement utilisé pour les échanges avec un module analoqique.

Modifiez le numéro d'esclave si nécessaire.

Commande : Si aucun bit n'est spécifié l'interrogation de l'équipement se fait cycliquement par la carte. Si un bit est spécifié l'interrogation de l'équipement concerné s'effectue lorsque ce bit (Boolean) est à 1. Lorsque la trame à été lue (l'équipement à répondu sans erreur) ce bit est remis à 0 (False) par la carte renvoyant ainsi un accusé de réception.

Diagnostique : A la sauvegarde, le configurateur crée les fichiers de communication PCARSICP.PAS et .VPU ainsi que les variables Globales de diagnostique suivantes :

```
STATUSICP : ARRAY[0..255] OF Boolean;
COMICPOK  : Boolean;
```

COMICPOK : Boolean -> vous informe sur l'état du port de communication utilisé
COMICPOK est à 1 (True) si le port s'est ouvert correctement
COMICPOK est à 0 (False) si il y a eu erreur d'ouverture du port utilisé.

STATUSICP : Array[0..255] Of Boolean -> vous informe sur l'état de l'échange de communication avec l'équipement respectif.
STATUSICP[5] est à 1 (True) si l'échange avec l'équipement n°5 s'est déroulé correctement.
STATUSICP[5] est à 0 (False) si l'échange avec l'équipement n°5 à échoué.

Utilisation des compteurs des modules de type D :

Les entrées Tor des modules de type D peuvent être utilisées comme compteur (100 Hz).

Modules 7041D, 7052D, 7053D, 7044D, 7050D, 7060D.

Lors de la déclaration de ces modules dans la grille de paramétrage, 2 lignes complémentaires apparaissent :

	Equipement	Description	Type	NB Bytes	Mot Visual PLC	Commande	C.Garde	Commentaires
1	1	7050D	Input	1			0	7xDI 8xDO
2	1	7050D	Output	1			0	7xDI 8xDO
3	1	7050D	Count	8	W1	B1	0	4xD/I can be used at counter
4	1	7050D	Clear	1	Reset	B2	0	Clear 4 counter $000F

La première ligne permet d'effectuer une lecture cyclique ou acyclique (par bit de commande) des 4 compteurs disponibles dans le module. Les valeurs sont transférées dans les quatre mots consécutifs de type Integer, à partir de la variable W1.

La deuxième ligne permet de faire une remise à zéro de ces compteurs. Chacun des quatre compteurs est associé à un bit du mot Reset (compteur1->bit0..compteur4->bit3). Dans l'exemple ci-dessus si reset vaut 15, les quatre compteurs seront remis à zéro sur commande du bit B2. Si reset vaut 9, le compteur 1 et le compteur 4 seront remis à zéro sur commande du bit B2.

Chien de garde

Il est possible d'affecter une valeur de chien de garde à chaque module comprenant des sorties Tor ou analogiques. Ce chien de garde a pour but de faire retomber les sorties du module si celui-ci est déconnecté du maître depuis un temps supérieur à la valeur du chien de garde.
La valeur du chien de garde est un multiple de 100 ms. Attention de correctement calibrer le chien de garde en fonction de la vitesse utilisée et du nombre d'esclaves connectés sur le réseau. Plus le nombre d'esclaves est important plus le temps de scrutation général s'allonge.

Mot Visual PLC	Commande	C.Garde	Commentaires
		10	7xDI 8xDO
		10	7xDI 8xDO
W1	B1	10	7xDI 8xDO
Reset	B2	10	7xDI 8xDO
		x100ms	

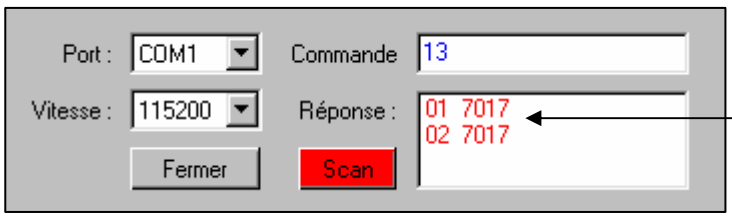
Chien de garde d'une seconde.

Chaque module ICP dispose d'une Led en face avant. Cette Led est allumée fixe quand le module est interrogé cycliquement par le maître. Elle clignote quand le module est déconnecté du maître depuis un temps supérieur à la valeur de chien de garde paramétré.

Led de diagnostique du module.

Ligne de commande et Scan

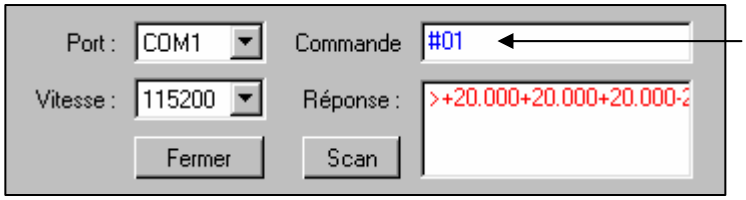
Le configurateur ICP permet de scanner par la liaison série, l'ensemble des modules connectés et paramétrés à la vitesse de communication du Scan.



Ici 2 équipements 7017 (1 et 2) sont connectés et paramétrés à 115 200 Bauds.

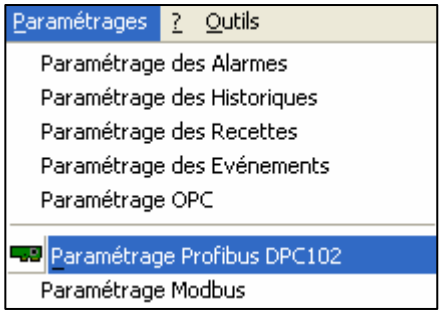
Il est également possible d'exécuter les différentes lignes de commande spécifiées dans les brochures livrées avec chacun des modules ICP. Ces commandes permettent le paramétrage des modules, la lecture des entrées Tor, la lecture des entrés analogiques, etc…

Exemple : Lecture des entrées analogiques de l'équipement 7017 n°1.



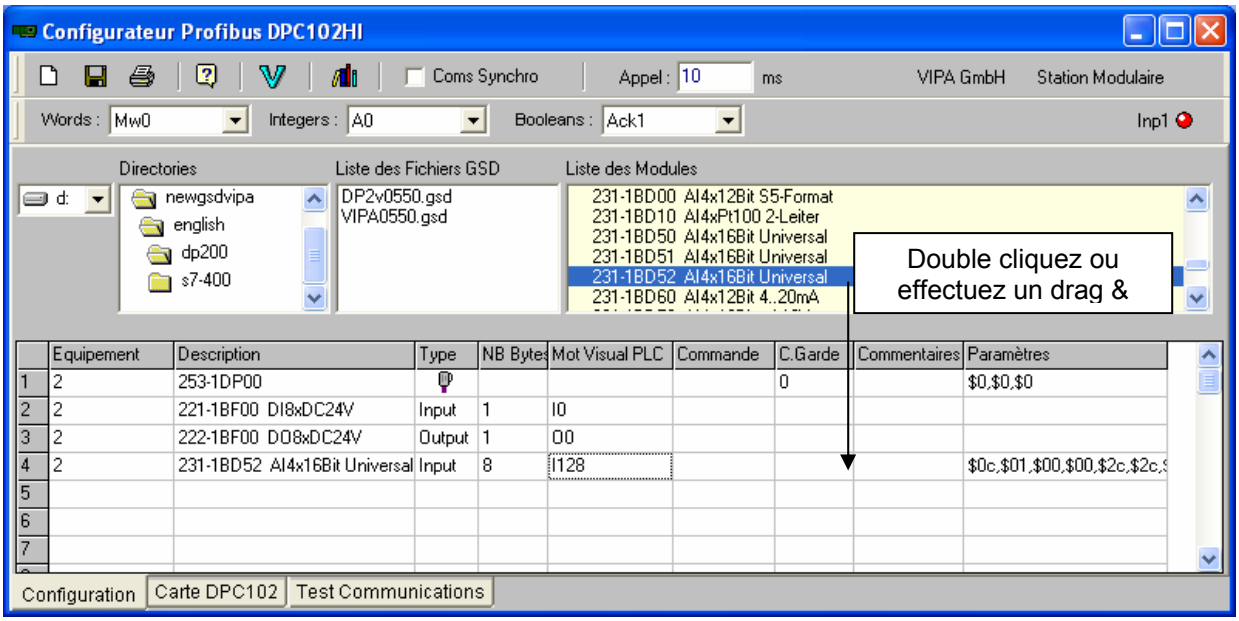
Envoi de la commande par la touche 'Entrée' du clavier.

Paramétrage PROFIBUS – Carte DPC102



Le configurateur est un programme externe à l'éditeur Visual I/O comme les autres configurateurs (sauf Modbus qui lui est intégré).

Le paramétrage PROFIBUS utilise des fichiers nécessaires de configuration appelés communément fichiers GSD.
Choisissez un fichier GSD (d'extension .GSD) par navigation sur votre disque dur.
La liste des modules apparaît dans la listbox de droite (Font jaune).
Amenez en premier par Drag & Drop ou double clic une tête de station puis le type de module souhaité dans la grille.



En fonction du type de modules choisi, le nombre de bytes en Input et output s'affiche. Renseignant ainsi l'utilisateur sur le choix de la variable Visual PLC (Ou visual I/O) à inscrire dans la colonne Mot Visual PLC.

Equipement : Frappez le numéro de l'équipement (Roue codeuses sur la station Profibus).
Description : Texte contenu dans le fichier GSD.
Type : INPUT (Entrées Profibus). OUTPUT (Sorties Profibus).
NB Bytes : Nombre d'Octets à lire (disponibles) dans l'équipement Profibus.

Mot Visual PLC (ou Visual I/O) : Premier mot de rangement des octets lus ou écrits dans l'équipement Profibus concerné.
Commande : Si aucun bit n'est spécifié l'interrogation de l'équipement se fait cycliquement par la carte. Si un bit Visual PLC est spécifié l'interrogation de l'équipement concerné s'effectue lorsque ce bit (Booléen) est à 1. Lorsque la trame à été lue (l'équipement a répondu sans erreur) ce bit est remis à 0 (False) par la carte renvoyant ainsi un accusé de réception.
C.Garde : Chien de garde (WatchDog). Si aucune valeur n'est spécifiée, les sorties Profibus reste dans l'état même si plus aucune trame n'est envoyée par la carte. Si une valeur est spécifiée (x100ms) une trame doit être envoyée cycliquement par la carte avant d'atteindre cette valeur. Si ce n'est pas le cas remise à zéro des sorties et passage en Bus Fail par l'équipement concerné. Valeur communément utilisée 20 (2 secondes)

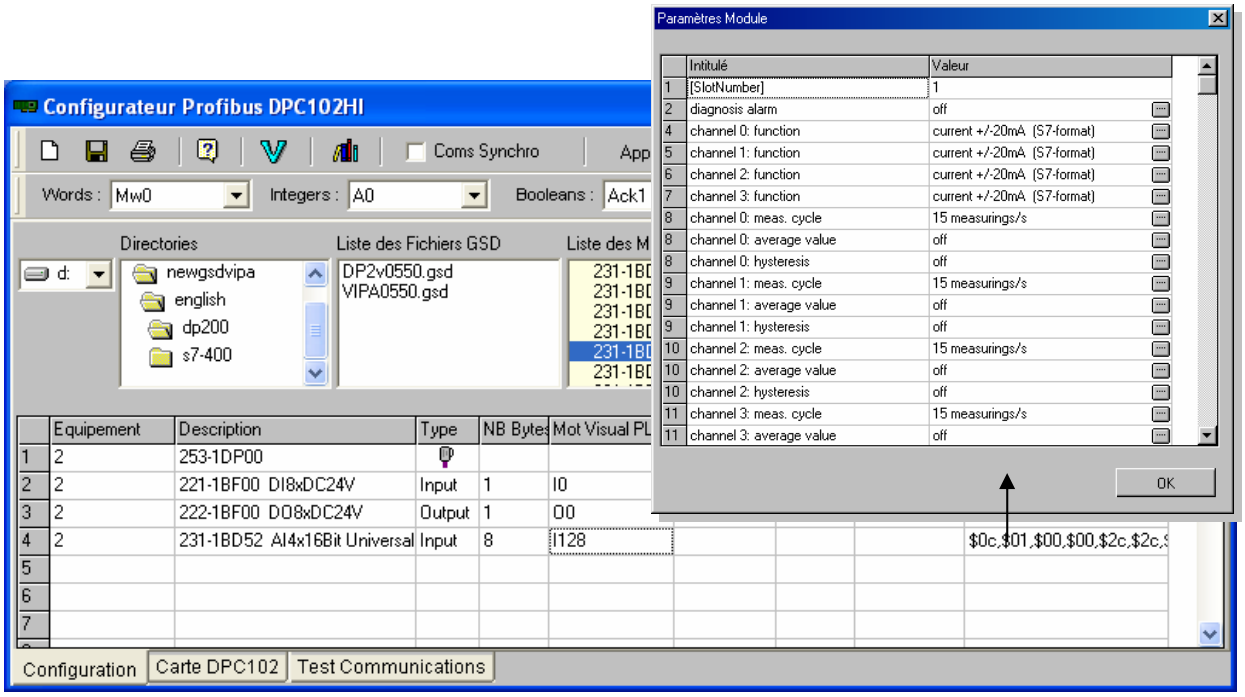
Commentaires : Vos commentaires.
Paramètres : Permet de visualiser ou de modifier les paramètres d'initialisation de la station et des modules.

☞ **Note** : **En double cliquant sur la cellule une boite de paramètres apparaît...**

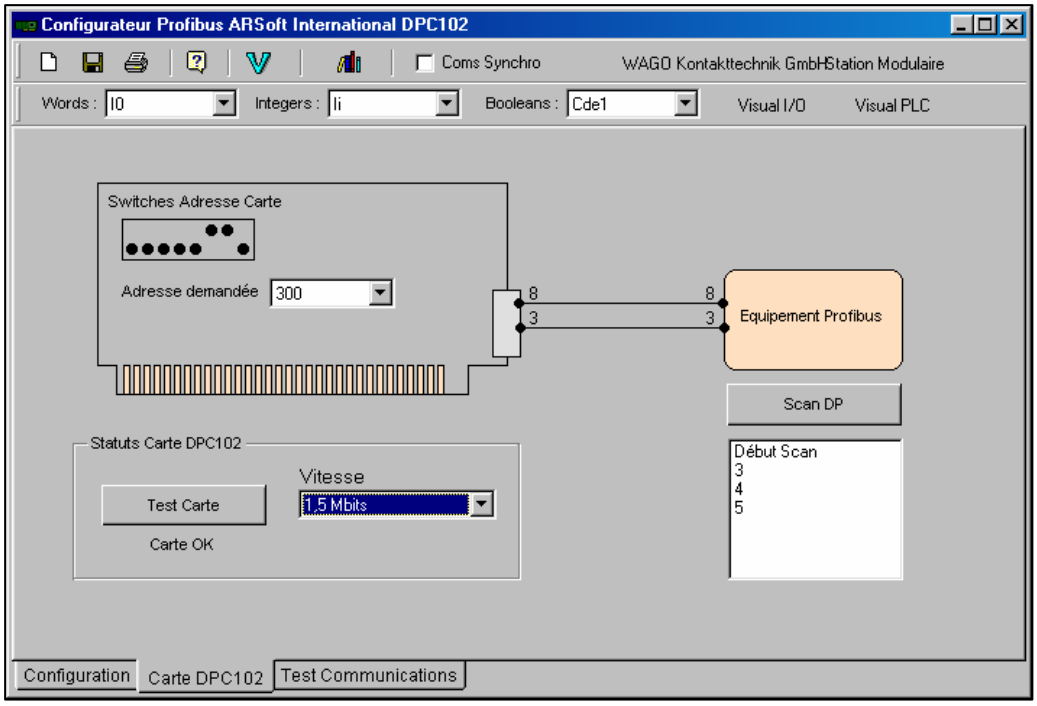
Case à cocher Coms Synchro : Si cette case est cochée, le driver **PCARSDP.VPU** généré est différent permettant **d'enchaîner** systématiquement toutes les trames Profibus paramétrées toutes les 10 ms. Si cette case à cocher est décochée, les trames Profibus seront envoyées une à une toutes les 10 ms.
Sauvez, le configurateur compile un driver par rapport à votre configuration. Le programme **PCARSDP.VPU** est à mettre dans le moteur temps réel de Visual PLC pour que la communication démarre.

Réglage des paramètres de l'esclave

Double-cliquez dans la **colonne paramètres** pour régler chaque paramètre du module concerné.
Ci-dessous un module de 4 entrées analogiques ou vous devez configurer chaque voie (4/20ma, 0/10V, PT100 etc..)



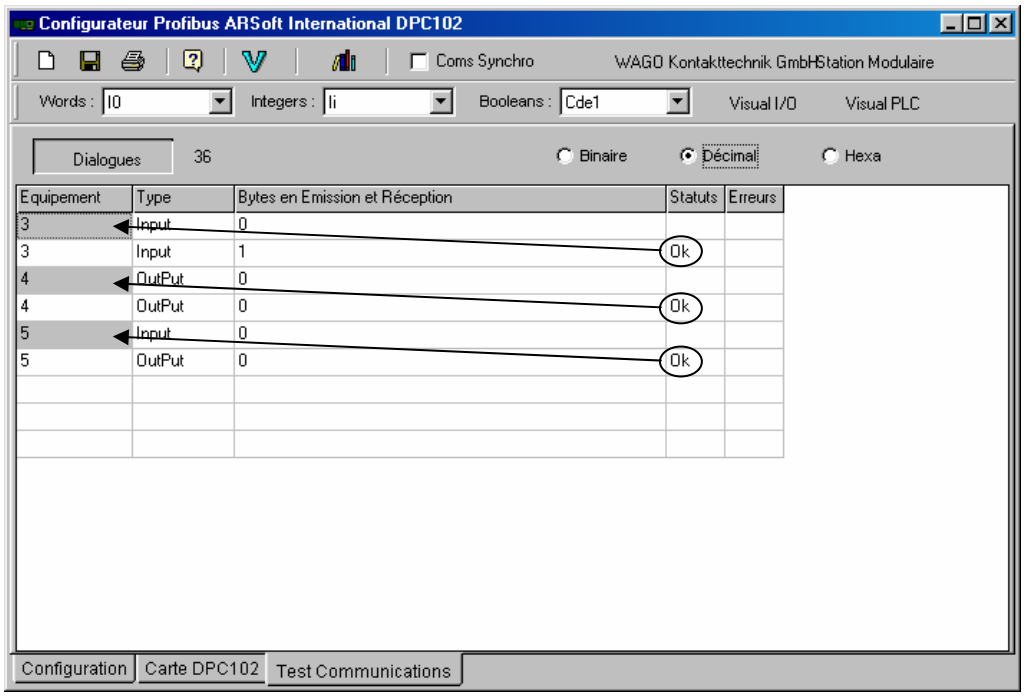
Test de la carte Profibus et ScanDP.



Cliquez sur le bouton **Test Carte** pour tester entièrement la carte Profibus (Mode « loop Back »)
Cette commande indique la position des Switches sur la carte si cette carte est au bus ISA ou PC104. Si cette carte est une carte au bus PCI, Windows effectue le plug & play et définie l'adresse de la carte automatiquement. Si la carte n'est pas détectée, il est probable que celle-ci soit en conflit d'adresse avec d'autres cartes sur le bus. Choisissez une adresse dans le Combo 'Adresse demandée', le système vous indique alors la nouvelle position des Switches.

Scan DP : Permet de scanner les équipements connectés sur le bus Profibus. Du fait de l'autoadaptation de la vitesse, les esclaves ne répondent pas forcément tout de suite.
Cette commande est à faire plusieurs fois.
Certains esclaves mémorisent en interne la dernière vitesse, il faut donc couper leur alimentation pour la changer.

Test de dialogue avec les équipements Profibus connectés



Cliquez sur le bouton **Dialogues**, une grille apparaît affichant l'état des octets d'échanges lus et écrits dans les équipements configurés. Vous pouvez alors forcer certains en affectant une nouvelle valeur dans la cellule concernée.

Les cellules grisées dans la colonne Equipement, indiquent le premier Octet de l'équipement.

Il est évident que l'on ne peut que forcer que les Octets 'OutPut'. Si les équipements répondent normalement, le texte '*Ok*' apparaît sinon le nombre d'erreurs détectées apparaît dans la colonne erreurs.

Ici le premier texte '*OK*' en partant du haut correspond à l'équipement N° 3.

Interactions dangereuses avec Visual PLC

Le configurateur PROFIBUS est un programme externe qui génère et compile une VPU qui est le driver Profibus lui-même. Ce driver **PCARSDP.VPU** est utilisable aussi bien par le moteur temps réel de Visual PLC que de Visual I/O. Dans le cas ou vous utilisez Visual PLC et Visual I/O ensemble il est impératif qu'il n'y est qu'un seul PCARDSP.VPU d'actif sur votre PC.

IL faut donc que PCARSDP.VPU soit dans les taches prioritaires de Visual I/O ou dans le moteur temps réel de Visual PLC MAIS PAS DANS LES DEUX.
Le plus souvent le driver Profibus relève de l'automatisme donc de Visual PLC.

Fichiers nécessaire au fonctionnement de la carte Profibus.

DPConf.DP : Le fichier de configuration (le contenue de la grille) dans le répertoire de l'appli.
ARSBT.SYS: le driver Windows. Dans le répertoire Windows/driver
Arsdp.dll : La DLL d'interface avec le fichier de configuration dans le répertoire Windows.

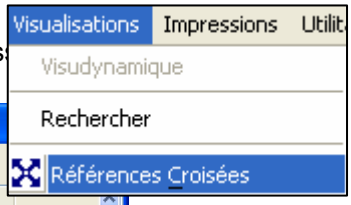
☞ **Note** : Dans le cas du panel PC Arsoft le fichier DPCONF.DP n'est plus nécessaire en run. Les paramètres sont inclus dans le driver Profibus.

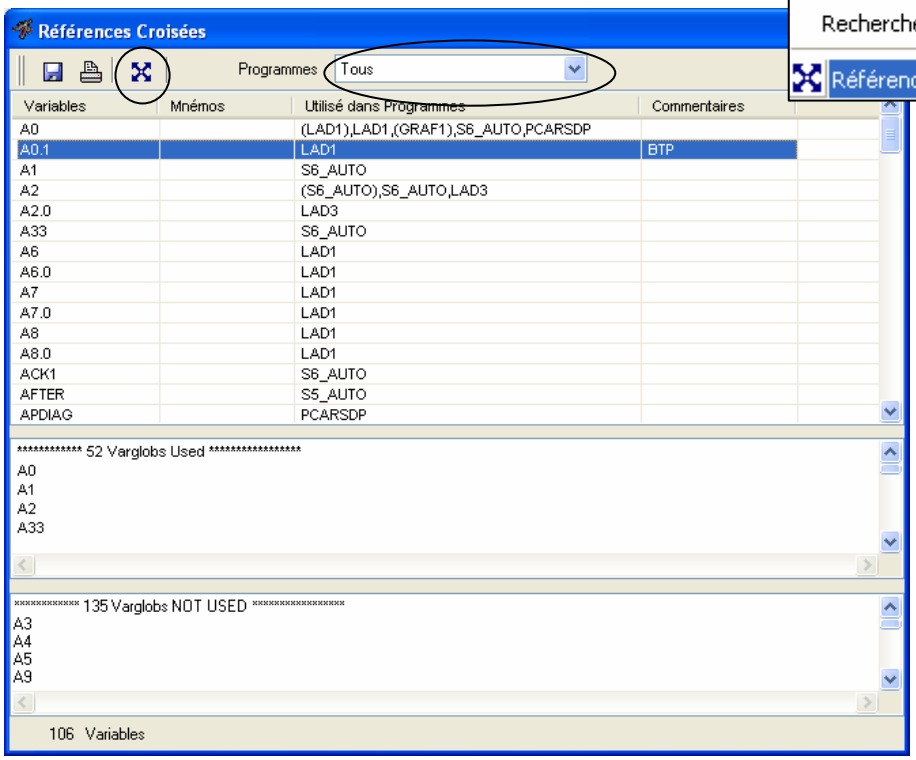
Coms synchro : Vous avez la possibilité de changer la fréquence d'activation du driver et de passer à 1 ms et multiples. Cependant considérant qu'un équipement Profibus réagit en 1 ms il est matériellement impossible de scanner à 1 ms plus d'un esclave Profibus.
Il faut si vous voulez interroger rapidement des esclaves Profibus passer le driver à 1 ou 2 ms et de ne pas activer Coms synchro.

Références croisées

Les références croisées sont générées à la compilation des synoptiques et des programmes. La compilation crée des fichiers .REF qui sont le résultat des variables trouvées par le compilateur.

Cliquez sur références croisées, une grille apparaît :
Choisissez un programme ou Tous puis cliquez sur le bouton ci-des





La grille supérieure se remplie dans la colonne variable qui dans le cas d'entiers (Byte, Word, integer) peut contenir un mot (A0) ou un bit de mot (A0.0)

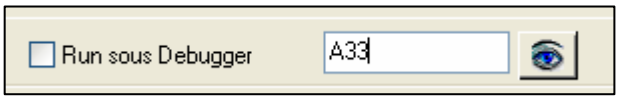
Mnémos : Ne sont pas utilisés par VisualI/O. Seul *Visual PLC* peut définir des mnémoniques pour les variables.
Commentaires : Ne sont pas utilisés par VisualI/O. Seul *Visual PLC* peut définir des commentaires pour les variables.

Utilisé dans programmes
Exemple A0 : (LAD1),LAD1,(GRAF1)

Signifie que A0 est affecté (en mot directement ou un bit du mot) c'est le programme LAD1 et GRAF1 qui écrive cette variable car le nom du programme est entre parenthèses.

Localisation d'une variable dans un synoptique.

Cette fonctionnalité n'est opérationnelle que sur des figures provenant des 2 premiers onglets



Frappez la variable recherché, puis cliquez le bouton à droite la figure ayant la variable demandée clignote sur le synoptique.

Deux listbox affichent les variables globales utilisées dans les programmes et celles non utilisées.

Composants Visual I/O

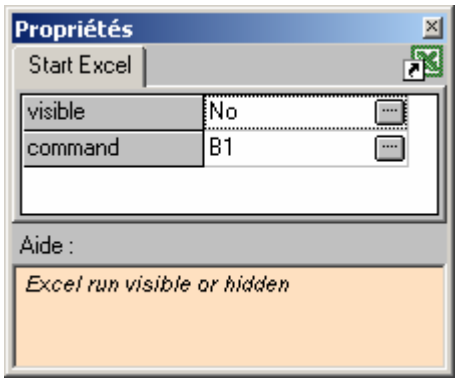
Onglet Excel

- Ouverture et Fermeture Excel
- Chargement d'une feuille excel.
- Création d'une feuille Excel
- Lecture et écriture dans cellules.

Les composants Excel sont au nombre de 9. Ils permettent la mise en place d'un lien OLE entre Visual I/O et Excel. Ils deviennent invisibles au lancement de votre application. Le dernier composant permet de constituer un fichier Excel à l'aide des valeurs de variables globales.

Composant StartXLS

Ce composant permet de lancer Excel visible ou caché depuis votre application.

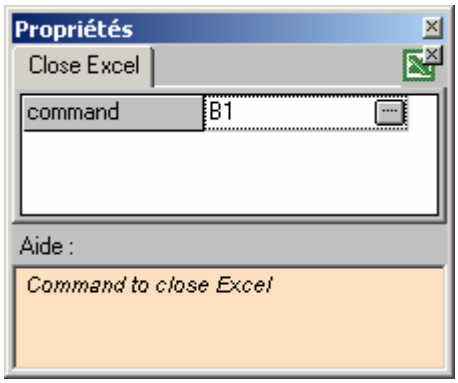


Visible : 'Yes' ou 'No' Lancement d'Excel visible ou caché.

Command : Variable de commande de lancement de l'application Excel.

Composant StopXLS

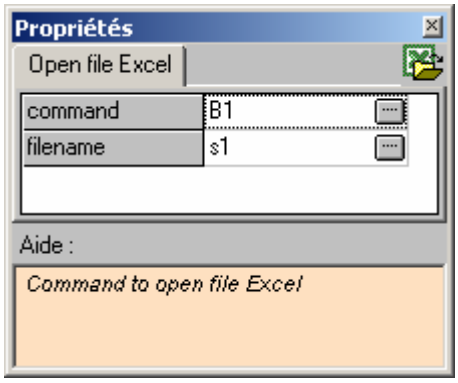
Ce composant permet de fermer Excel depuis votre application.



Command : Variable de commande de fermeture de l'application Excel.

Composant OpenXLS

Ce composant permet d'ouvrir un fichier Excel depuis votre application.

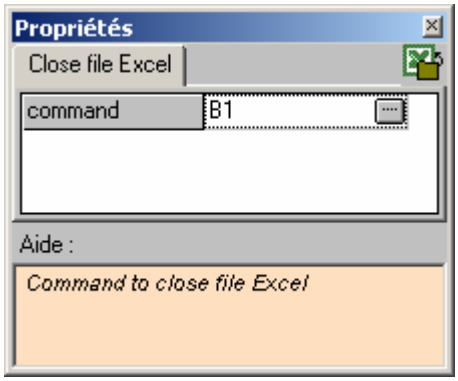


Command : Variable de commande d'ouverture de fichier.

FileName : Nom du fichier XLS à ouvrir.
Si une variable de type String vide ('') est passée en paramètre au composant, une boite de dialogue Windows d'ouverture de fichier apparaît. La variable prend alors le nom du fichier sélectionné dans la boite de dialogue Windows.

Composant CloseXLS

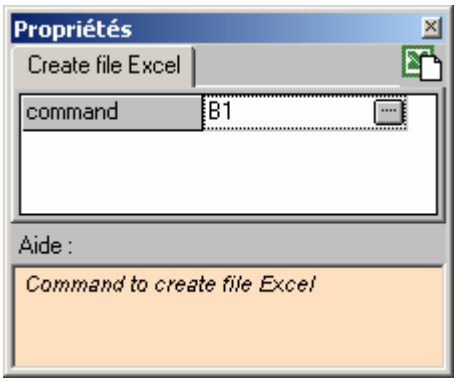
Ce composant permet de fermer un fichier Excel depuis votre application.



Command : Variable de commande de fermeture de fichier.

Composant NewXLS

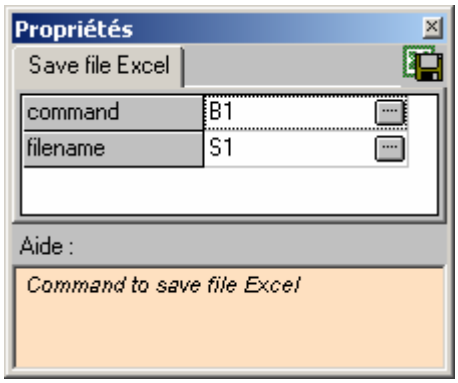
Ce composant permet de créer une grille Excel vierge depuis votre application.



Command : Variable de commande de création d'une grille Excel vierge.

Composant SaveXLS

Ce composant permet d'enregistrer le fichier Excel ouvert depuis votre application.

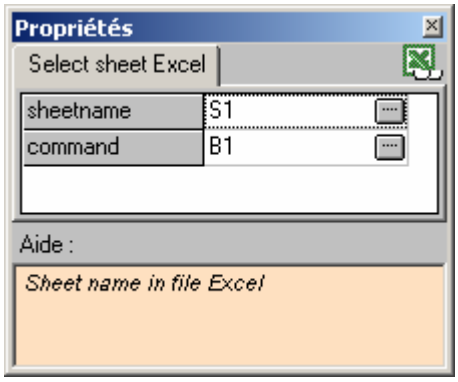


Command : Variable de commande d'enregistrement du fichier ouvert.

FileName : Nom de sauvegarde du fichier XLS. Si une variable de type String vide ('') est passée en paramètre au composant, une boite de dialogue Windows d'enregistrement de fichier apparaît. La variable prend alors le nom du fichier sélectionné dans la boite de dialogue Windows.

Composant SheetXLS

Ce composant permet de sélectionner une feuille du fichier Excel ouvert depuis votre application.

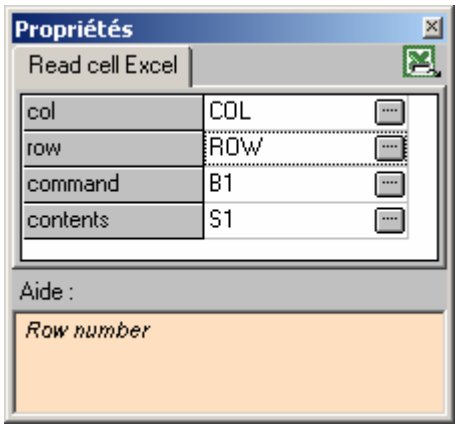


SheetName : Nom de la feuille du fichier ouvert à sélectionner.

Command : Variable de commande de sélection de feuille.

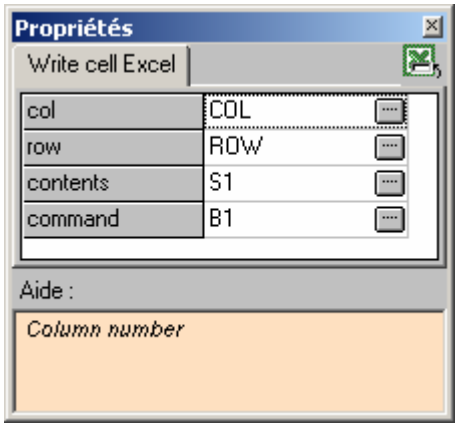
Composant ReadXLS

Ce composant permet de lire le contenu d'une cellule Excel depuis votre application.



Composant WriteXLS

Ce composant permet d'écrire dans une cellule Excel depuis votre application.



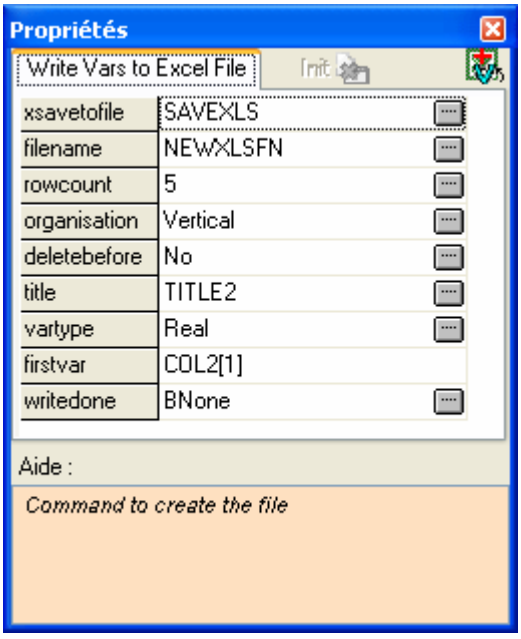
Col : Numéro de colonne de la cellule à écrire.
Row : Numéro de rangée de la cellule à écrire.

Command : Variable de commande d'écriture dans la cellule.

Contents : Variable de type String contenant le texte à écrire dans la cellule.

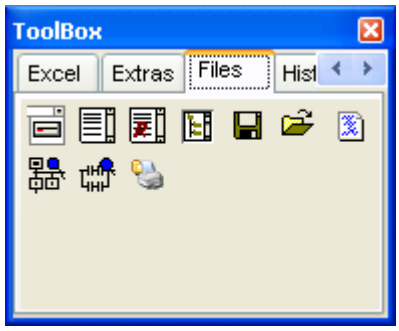
Composant ColVarToXLS

Ce composant permet de remplir une colonne ou une rangée dans un fichier exploitable par Excel avec les valeurs de variables globales consécutives (il faut simplement spécifier la première variable, son type et le nombre de variables consécutives à celle-ci). Afin de constituer plusieurs colonnes dans un fichier XLS ou CSV vous devez utiliser plusieurs composants ColVarToXLS. Chaque composant traite une colonne complète avec des variables globales. Si le fichier Excel doit contenir 5 colonnes il faut utiliser 5 composants distincts.



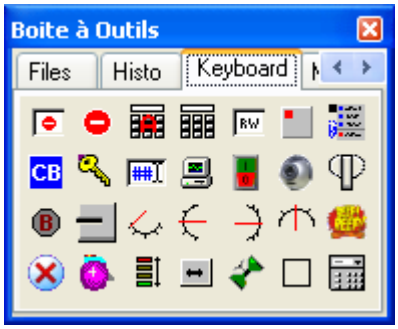
XSaveToFile : Commande pour sauvegarder dans le fichier de nom spécifié dans **FileName**.
Rowcount : Nombre de cellules à écrire avec des variables conécutives ceci soit en vertical ou en horizontal.
Organisation : Ecriture dans une colonne (Vertical) ou dans une rangée (Horizontal).
DeleteBefore : Yes permet d'effacer le fichier final avant le traitement. Fixez Yes pour la première colonne ou rangée.
Title : Titre de la colonne ou de la rangée. Si chaine vide aucun titre n'est ajouté à celle-ci.
Vartype : Type de la série de variables à afficher dans la colonne ou rangée.
FirstVar : Nom de la première variable à afficher dans la colonne ou la rangée. Les autres variables sont celles déclarées consécutivement dans les variables globales.
WriteDone : Bit optionel passant à true quand l'écriture est effectuée.

Onglet Files



- Choix d'un lecteur de disques.
- Liste de fichiers à extension donnée.
- Liste des répertoires
- Visualisation d'un fichier texte.
- Choix pour sauvegarde dans fichier.
- Choix pour chargement d'un fichier.
- Visualisation d'un Grafcet
- Visualisation d'un Ladder
- Réglages Imprimante.

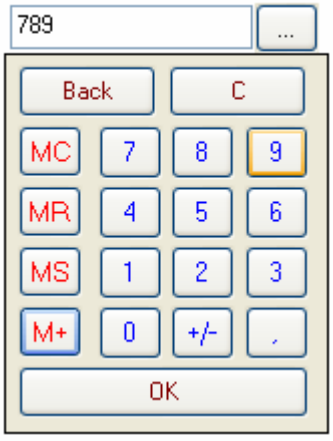
Onglet Keyboard



- Saisie de mot de passe
- Saisie de mot de passe
- Clavier Alphanumérique
- Clavier Numérique.
- Zone de saisie en lecture/écriture
- Bouton design.
- Neutralisation affichage bureau.
- Bouton de 2 couleurs
- Gestion de Niveaux d'accès.
- Zone de saisie filtrée.
- Neutralisation Ctrl/alt/del
- Interrupteur.
- Interface pour Webcam..
- Interrupteur rotatif 3 positions.
- Petite zone type bouton
- Potentiomètre Vertical.
- 4 Gauges
- Marche / Arrêt buzzer du PC (sur Carte mère)
- Bouton bitmap de fermeture du synoptique.
- Animation par bitmaps successifs.
- Bargraph à 3 niveaux de couleurs
- Zone de déplacement de la fenêtre du synoptique.
- Affichage de plusieurs images en fonction de bits.
- Zone de clic souris.
- Zone de saisie avec clavier numérique tactile.

Composant Zone de saisie avec clavier numérique

Ce composant permet d'afficher et de saisir une valeur numérique (Integer ou real). En cliquant sur le bouton à droite de la zone de saisie, un clavier numérique apparaît en dessous permettant ainsi de saisir la valeur numérique à l'aide par exemple d'un écran tactile. Les touches MS et MR sont Memory Clear (efface la mémoire interne) et MR Memory Recall (rappelle la valeur de la mémoire) La touche M+ permet d'additionner la valeur saisie à la mémoire interne du composant.
La touche MS permet de mettre en mémoire la valeur en cours.

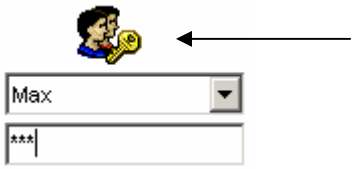


Si vous désirez modifier le composant, celui-ci utilise le synoptique NUMCLAV.SYN qui est dans le sous répertoire modules (C:\API32\Modules). Ramener alors tous les fichiers commencant Par NUMCLAV (NUMCLAV*.*) dans votre répertoire de travail.

Modifiez le clavier comme un synoptique normal puis renvoyez Tous les fichiers (numclav*.*) dans le sous répertoire Modules. Recompilez alors le composant (click droit/Création composant). Pour une validation de la valeur sur click du bouton OK, fixez l'attribut OnReturnOnly. Si vous voulez une validation immédiate fixez l'attribut Always.

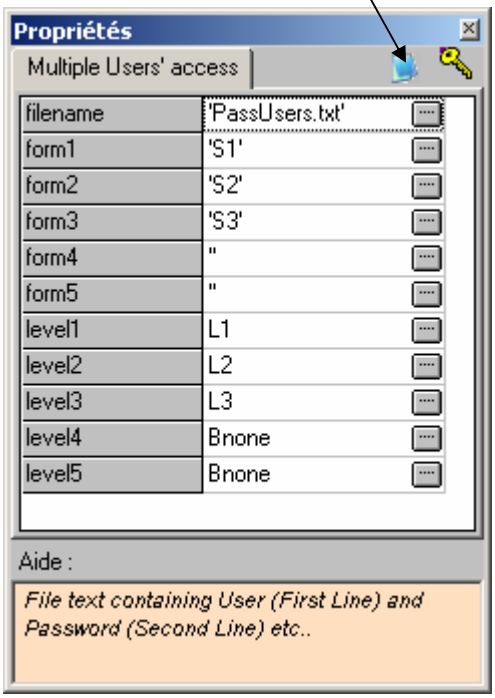
Composant UserAccess

Le composant User permet la mise en place de différents niveaux d'accès dans votre application. Les niveaux d'accès et les mots de passe correspondant sont stockés dans un fichier texte. Chacun des niveaux est associé à une variable booléenne *Level* permettant de rendre visible ou caché certaines parties de vos synoptiques ou d'interdire l'accès à certains menus. Chacun des niveaux est associés à une variable *Form* du type String qui contient le nom d'un synoptique appelé.



Un double clic fait apparaître un Combo Box permettant de choisir le type d'utilisateur et une EditBox pour saisir le mot de passe correspondant.

Cliquez ici pour appeler le notepad afin de déclarer le nom des utilisateurs ou niveaux et les mots de passe associés.



Le fichier texte contenant les utilisateurs et les mots de passe est définit de la manière suivante :
La première ligne définie l'utilisateur sélectionné au lancement de l'application. La deuxième ligne définie le mot de passe du premier utilisateur. La troisième ligne définie le deuxième utilisateur. La quatrième ligne définie le mot de passe du deuxième utilisateur. Etc.…
Il est possible de déclarer au maximum 5 utilisateurs.

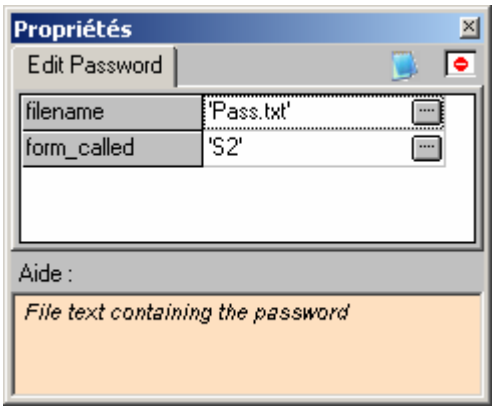
Exemple de fichier texte utilisateur:
Maintenance
123
Opérateur
456

FileName : Nom du fichier contenant les différents utilisateurs et mots de passe.
FormX : Nom du synoptique appelé après la saisie du mot de passe correspondant à l'utilisateur X définit dans le fichier.

LevelX : Variable Booléenne positionnée à True indiquant l'utilisateur après saisie du mot de passe correspondant à celui du fichier.

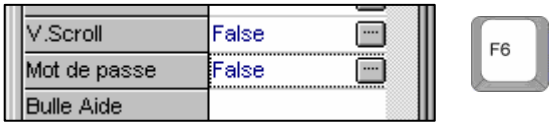
Composant EditPassword

Le composant EditPassword permet la mise en place d'un mot de passe pour accéder à un synoptique.
Le mot de passe est stocké dans la première ligne d'un fichier texte.



FileName : Nom du fichier contenant le mot de passe.

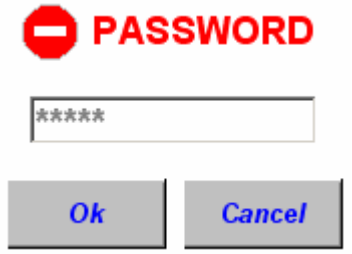
Form_Called : Synoptique appelé après validation du mot de passe par la touche 'Entrée'.



Il est possible dans les propriétés de la zone Edit <F6>, de retirer le masquage du mot de passe par des astérisques.

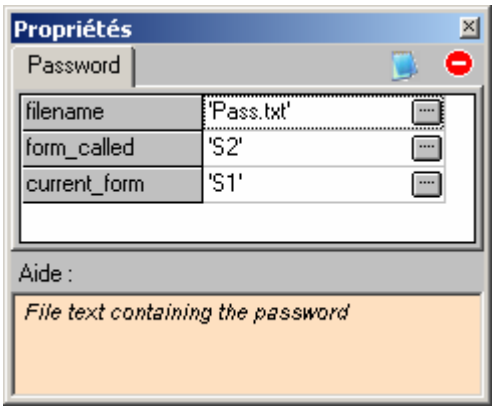
Composant Password

Le composant Password permet la mise en place d'un mot de passe pour accéder à un synoptique.
Le mot de passe est stocké dans la première ligne d'un fichier texte.
Ce composant est composé d'une zone de saisie du mot de passe, d'un bouton de validation et d'un bouton de sortie.



Ce composant est prévu pour être placé dans un synoptique qui apparaîtra lors d'une demande d'accès à une partie protégée de l'application.

Exemple : `If Button Then Display('MotDePasse') ;`



FileName : Nom du fichier contenant le mot de passe.

Form_Called : Synoptique appelé après validation du mot de passe par le bouton 'Ok'.

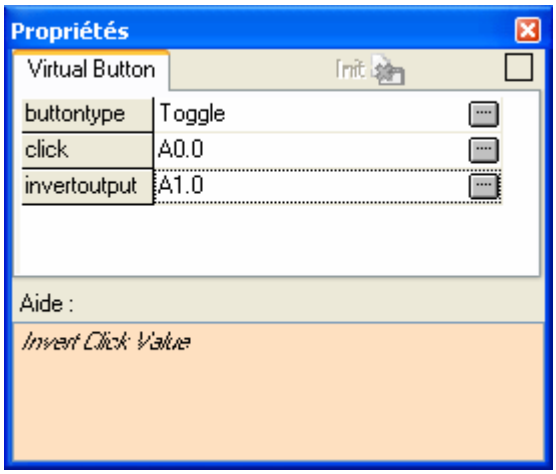
Current_Form : Synoptique contenant le composant Password. Ce synoptique est automatiquement fermé après validation du mot de passe par le bouton 'Ok' ou sur abandon de l'opérateur par le bouton 'Cancel'.

Composant Déplace le synoptique

Ce composant permet de déplacer le synoptique en cliquant dans la zone puis le déplacer à l'aide de la souris. L'action à effectuer est similaire à celle faite à l'aide de la barre de titre du synoptique.

Composant Bouton virtuel

Ce composant d'afficher un cadre (rectangle) permettant à l'opérateur de cliquer dessus afin de s'en servir comme d'un bouton classique. Ce composant permet alors de transformer n'importe quel zone de l'écran en bouton.



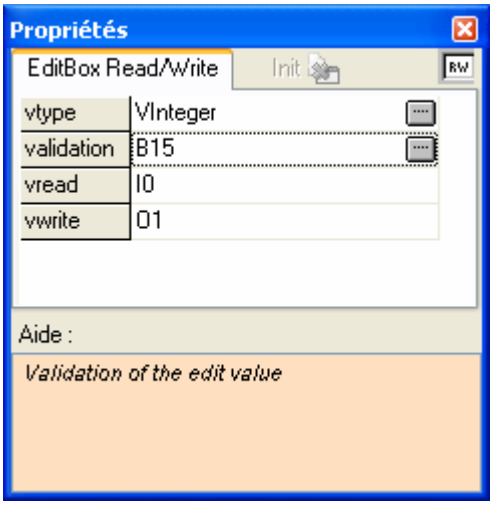
ButtonType : Simple ou accrochage. (Simple ou toggle).

Click : Booléen affecté en cas de click ou action sur la dalle tactile.

InvertOutput : Boolean état inverse de la sortie click.

Composant EditRW

Le composant EditRW permet d'utiliser une EditBox en lecture et en écriture. Cet EditBox peut-être utilisée avec les différents types de variables de Visual I/O. Lecture et écriture peuvent être associées à deux variables différentes



VType : Permet de définir le type de variables affectées par l'EditBox. (VWord, VReal, VString, etc…).

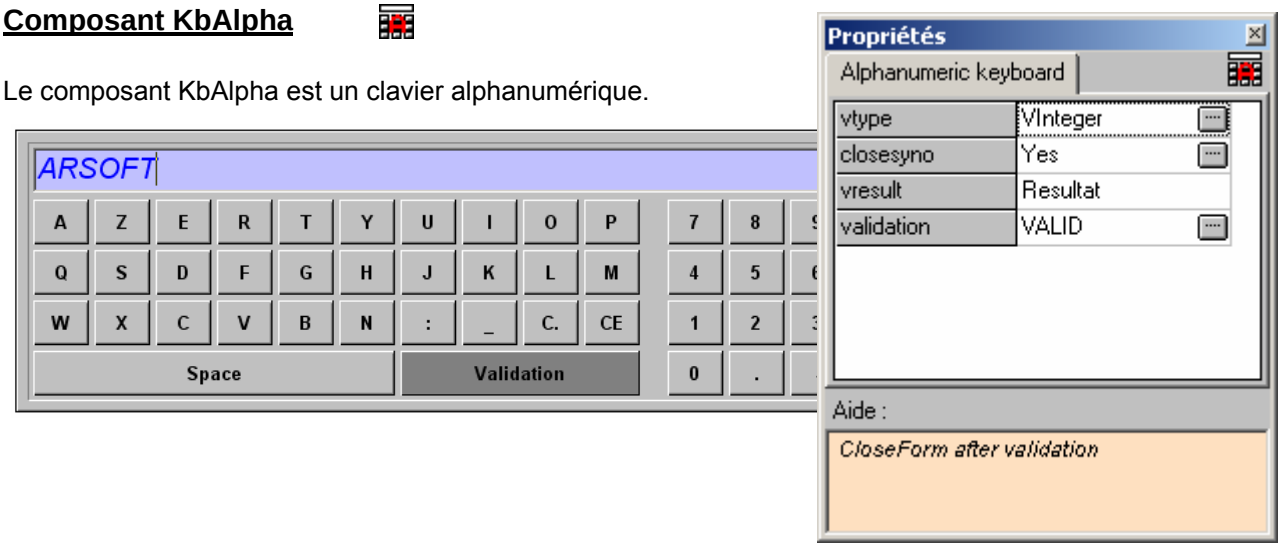
VRead : Variable visualisée dans l'EditBox (Lecture).
VWrite : Variable affectée par l'EditBox (Ecriture).

Validation : Variable de validation de la saisie dans l'EditBox.

La validation du contenu de l'EditBox est faite quand le Focus quitte le champ de saisie ou quand l'opérateur frappe la touche <Entrée> ou le bit validation passe à True.
☞ **Note** : Le réaffichage de la valeur contenue dans le paramètre VRead ne se fait que lorsque cette variable change de valeur ou à l'affichage du synoptique (sur winapparition).

Composant KbAlpha

Le composant KbAlpha est un clavier alphanumérique.



VType : Permet de définir le type de variable utilisé par le clavier. (VWord, VReal, VString, etc…).

CloseSyno : Après validation, permet de fermer le synoptique qui contient le clavier alphanumérique.

VResult : Variable qui va recevoir le résultat de la saisie du clavier alphanumérique.

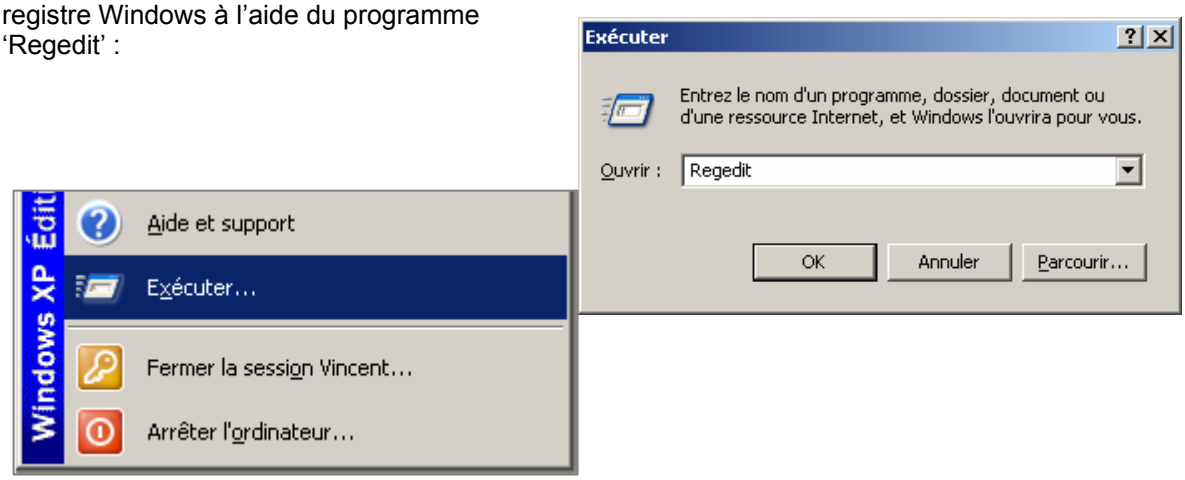
Validation : L'appui sur la touche 'Validation' du clavier effectue un Set de la variable 'Validation'. Le Reset de celle-ci doit être réalisé par programme.

Composant KillExplo

Le composant KillExplo permet de faire disparaître l'Explorer Windows (Bureau) au lancement de votre application afin d'interdire à tout utilisateur mal intentionné d'accéder à l'ordinateur. Le paramètre Reload permet de relancer ou non l'Explorer à la fermeture de votre application.

Reload : Ce paramètre permet d'autoriser ou non le redémarrage de l'Explorer Windows à la fermeture de votre application. Si le paramètre Reload est égal à 'True' ou si la variable BNone est associée à ce paramètre, l'Explorer Windows est relancé à la fermeture de l'application.

Attention : Le système d'exploitation Windows est paramétré de telle sorte que l'Explorer se relance automatiquement en cas de fermeture. Il est donc nécessaire de modifier la clé ci-dessous dans la base de registre Windows à l'aide du programme 'Regedit' :




```
HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\WINDOWS NT\CURRENTVERSION\
WINLOGON\AutoRestartShell = 0
```

Le composant KillExplorer fait appel à l'exécutable **KillExplo.exe**, celui-ci doit être présent dans le répertoire de votre application ou dans le répertoire C:\Windows ou dans le répertoire C:\Windows\System32.

Composant Ctrl_Alt_Del

Le composant Ctrl_Alt_Del permet d'interdire la combinaison de touches donnant l'accès au gestionnaire de tâches de
Windows pendant le fonctionnement de votre application. Ce composant ne nécessite aucun paramètre. Il devient invisible au lancement de l'application.

Attention
Ce composant doit être placé sur le premier synoptique affiché au lancement de votre application afin d'être exécuté dés le premier tour de cycle.

Doit être configuré dans la base de registre (lancez Regedit) :

```
HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\
WINDOWS NT\CURRENTVERSION\WINLOGON\AutoRestartShell = 0
```

Onglet mini recettes

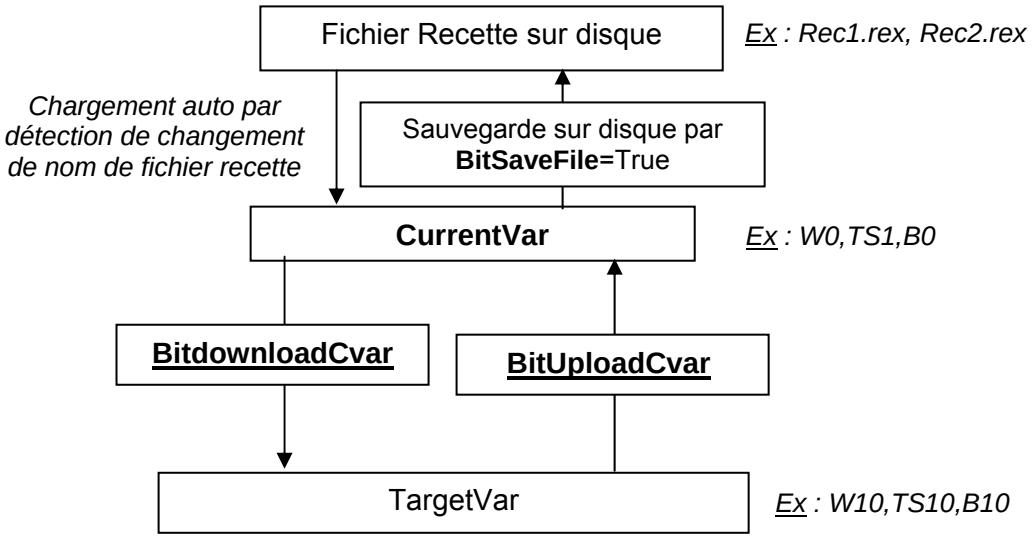
- Champs de saisie numérique et alphanumérique.
- Radio bouton
- Case à cocher
- Images
- Gestion de la navigation dans les fichiers.

Ces composants permettent la sauvegarde dans un fichier de valeurs numériques, alphanumériques, booléennes (radio boutons, case à cocher) et de nom d'image. Ceci permet ainsi de créer une recette par nom de fichier affectant des variables du process. La sélection d'une recette pourra se faire simplement par sélection d'un nom de fichier dans une liste.

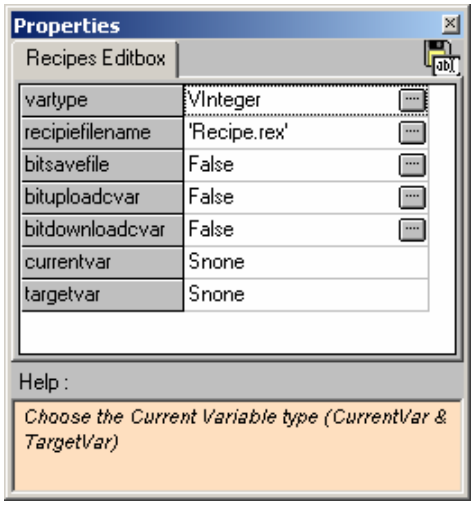
Principe

Nous travaillons sur une variable qui peut être visualisée et modifiée. Cette variable peut être la variable process elle-même ou une variable intermédiaire qui sera affectée au process par un mécanisme (**Download**) ou remontée du process par le mécanisme (**Upload**).
Dans le cas d'une descente de valeur depuis un fichier, la valeur prend le chemin suivant :
Transfert du fichier vers une variable courante puis si le bit download est à true cette valeur est aussi affectée à la variable finale process.

Dans le cas d'une remontée de valeur jusqu'à un fichier :
La valeur est lue dans le process (variable process) puis transmise dans une variable courante qui elle-même est sauvegardée dans un fichier.



Editbox pour recette



Saisie et visualisation de variable numérique ou string.

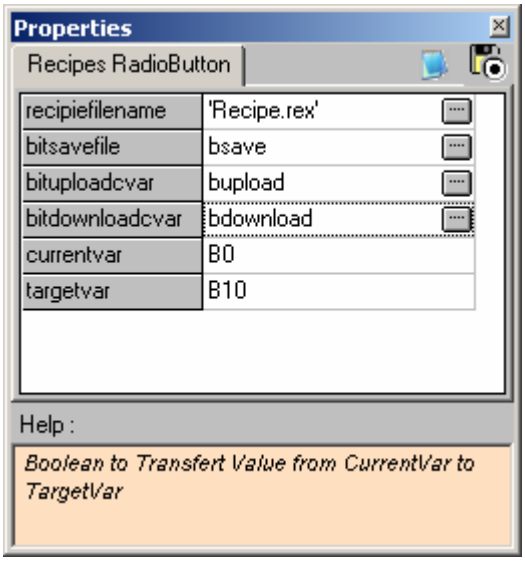
Vartype : Type de variable Visual I/O à affecter.

RecipeFileName : Nom ou variable contenant le nom du fichier de sauvegarde ou de chargement. Si ce nom change le champ de saisi est mis à jour automatiquement.
BitSave : Boolean pour sauver la valeur de currentVar dans le fichier.

BitUploadCvar : Permet de remonter la valeur de targetvar dans la variable currentvar.
BitdownloadCvar : Permet de faire descendre la valeur de la variable CurrentVar dans targetvar.

CurrentVar : Variable de travail de type Vartype.
Targetvar : Permet de recevoir ou de remonter des valeurs

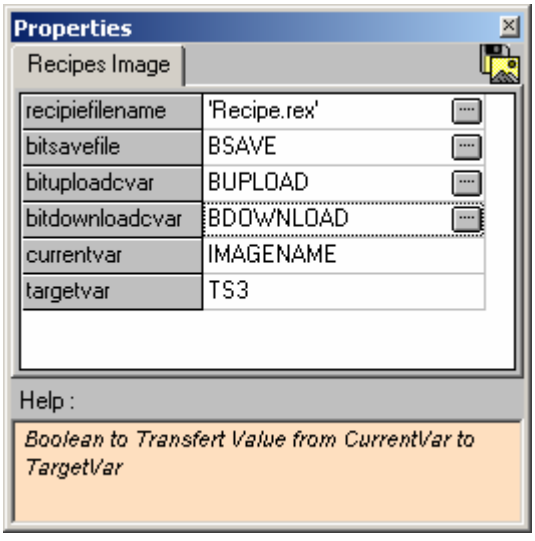
Radiobouton et case à cocher pour recette



Composant fonctionnant à l'identique par rapport aux autres.

Currentvar et targetvar doivent être de type booléen.

Image pour recette



CurrentVar contient le nom de l'image à afficher. Ce nom peut contenir le chemin complet de l'image.

Ex C:\mybmp\Photo1.bmp

☞ **Note** : Ces composants permettent de réaliser des écrans simples de saisie et de chargement de valeurs.
Ces recettes ne fonctionnent que si l'écran contenant ces composants est affiché. Il est impossible de télécharger ou de sauvegarder par programme les valeurs.
Des écrans type base de données peuvent être constitués

Les autres composants

FileList Renvoi la liste des fichiers d'une extension spécifiée sans possibilité de rafraîchir cette liste

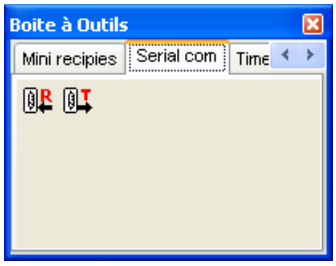
FileList with refresh Renvoi la liste des fichiers d'une extension spécifiée avec possibilité de rafraîchir cette liste avec un bit.

Directory listbox : Boîte liste des répertoires. Affiche la structure des répertoires du lecteur actif. Les utilisateurs peuvent changer de répertoire dans une boîte liste des répertoires

Drive combo box : Boîte à options des lecteurs. Affiche une liste déroulante des lecteurs disponibles.

Une illustration de ces composants est faite dans la rubrique *: Visualiser un fichier texte dans un champ de saisie*

Onglet liaison série



- Réception de caractères.
- Emission de caractères.

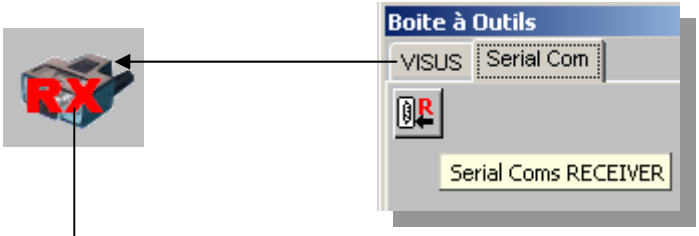
2 composants permettant simplement de recevoir ou d'émettre des caractères sur une liaison série. Plus simplement utilisable que les fonctions de la librairie serial .vpu.

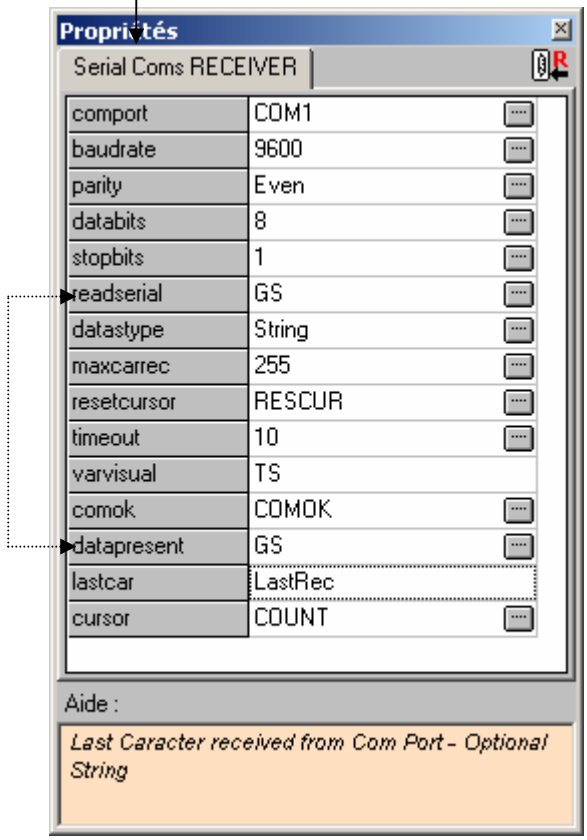
Serial Coms Receiver

Description du composant

- Initialisation de la liaison série COM1 à COM8.
- Réception des caractères et stockage dans une variable Visual I/O.
- Gestion de perte de communication (Time Out).

Attention : Pour que le composant soit opérationnel, le synoptique le contenant doit être visible. Si ce composant doit être toujours en service le mettre dans le synoptique principal de l'application





Configuration de la liaison série

Comport : Com1 → Com8
Baudrate : 110 → 115200 bauds
Parity : Even (Paire), Odd (Impaire), None (Aucune)
DataBits : 8 Bits ou 7 Bits de données
Stop Bits : 1 ou 2 bit de stop

Fonctionnalités

ReadSerial : Entrée booléenne permettant de lire et de stocker les caractères reçus sur la liaison série. En connectant cette entrée sur la sortie **Datapresent** une lecture est effectuée à chaque réception de nouveau caractère.

DataType : String ou Byte. Décrit si les caractères sont ASCII ou numériques.

ResetCursor : Permet de remettre à 0 le pointeur de réception.

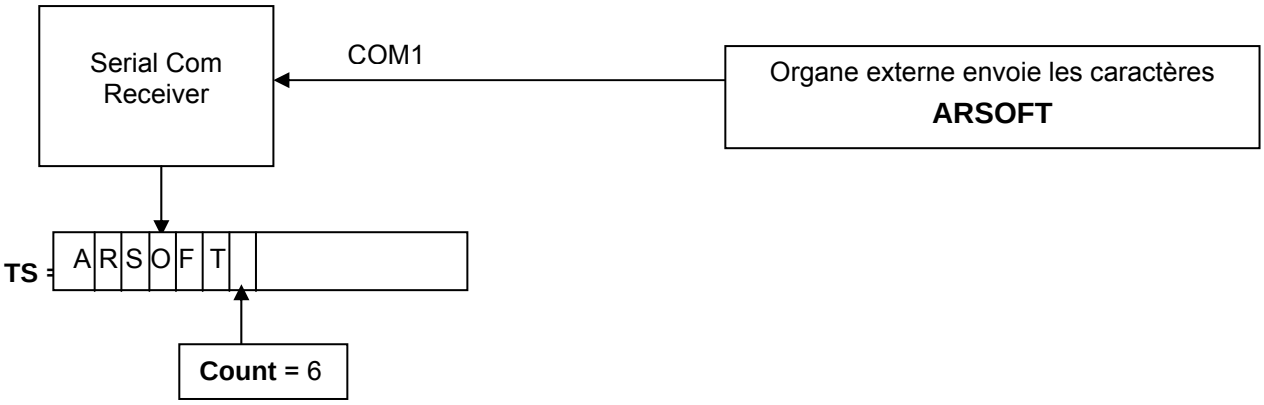
TimeOut : En x10ms permet de fixer un temps maxi entre 2 réception de caractères. Si ce temps est dépassé une remise à zéro du curseur et de la variable Visual I/O est effectuée. Si time out=0 alors aucun traitement n'est effectué.

VarVisual : Variable Visual I/O de type String ou Array of Byte.
Recevant tous les caractères reçus de la liaison série.
ComOk : Boolean passant à True si le port de communication s'est bien ouvert au lancement de l'application.
DataPresent : Bit passant à true si un caractère non lu est présent dans le buffer du port de com (buffer Windows)
LastCar : Byte indiquant le dernier caractère reçu.
Cursor : Curseur interne comptant le nombre de caractères dans la variable Visual I/O

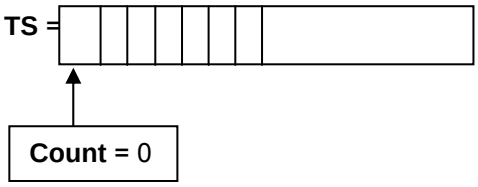
Exemple Variable Globales

```
GS          : Boolean;
ResCur      : Boolean;
TS          : String;
ComOK       : Boolean;
LastRec     : Word;
Count       : Integer;
```

Buffer Visual I/O



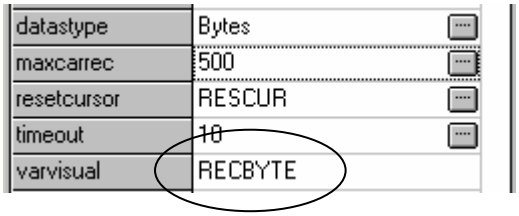
SI ResetCursor = True ou Timeout>0 ou Cursor>=255



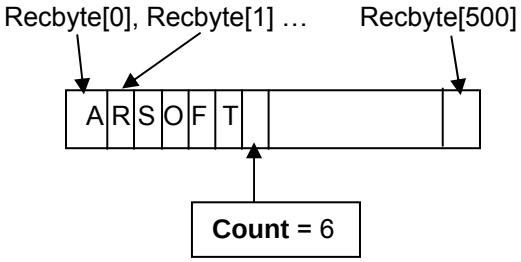
Le principe est le même avec un tableau de byte

Déclarez un tableau dans les Variables globales

RecByte : Array [0..500] Of Byte;



Ici **DatasType** est paramétré en réception de bytes et la variable Visual I/O est un tableau recevant les caractères de la liaison série.
Le maximum est fixé à la taille du tableau (ici dans notre exemple 500 bytes)



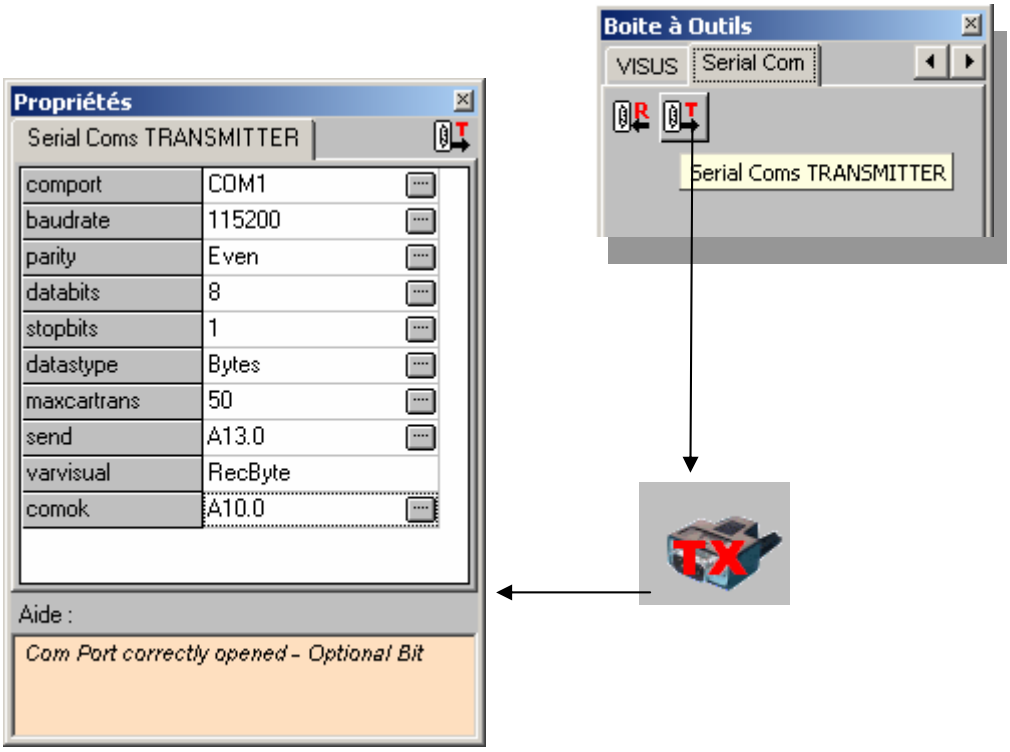
Serial Coms Transmitter

Nom du Fichier Com232TX.VPU

Description du composant

- Initialisation de la liaison série COM1 à COM8.
- Emission de caractères sur une liaison série.

Attention : Pour que le composant soit opérationnel, le synoptique le contenant doit être visible. Si ce composant doit être toujours en service le mettre dans le synoptique principal de l'application



Configuration de la liaison série

Comport : Com1 → Com8
Baudrate : 110 → 115200 bauds
Parity : Even (Paire), Odd (Impaire), None (Aucune)
DataBits : 8 Bits ou 7 Bits de données
Stop Bits : 1 ou 2 bit de stop

Fonctionnalités :
DataType : String ou Byte. Décrit si les caractères à envoyer sont ASCII ou numériques.

MaxCarTrans : Nombre de caractères à transmettre sur la liaison série. Ce paramètre est utile seulement dans le cas de transfert de bytes. Dans le cas de transfert de chaînes de caractères la longueur des chaînes est prise en compte pour l'émission.

Send : Commande permettant d'envoyer les caractères sur la liaison série. L'envoi se fait sur front montant, le programme doit remettre le bit à false et le repasser à true pour effectuer un nouvel envoi..

VarVisual : Variable Visual I/O de type String ou Array of Byte transmise sur la liaison série.

ComOk : Booléen passant à True si le port de communication s'est bien ouvert au lancement de l'application.

Instructions pour le pilotage de la liaison série – Librairie Serial.VPU

Vous pouvez aussi directement piloter la liaison série en utilisant les fonctions et procédures de la librairie Serial.VPU

```
// Ouvre un canal série
Function OpenCom (ComPort,Baudrate,Parity,Bits,Stop:String ): Boolean;

//  Test si le Canal série spécifié est déjà ouvert
Function  GetComOpen(Comport : Integer) : Boolean;

// Ferme un canal série
Procedure DisableCom (ComPort  : Byte);

// Ferme tous les canaux série
Procedure DisableAllComs;

// Test si un caractère est présent dans le buffer de réception
Function V24DataOk (ComPort  : Byte) : Boolean;

// Récupère un caractère dans le buffer de réception
Function GetByte (ComPort  : Byte ) : Byte;

//  Envoie une chaîne de caractères dans le buffer d'émission du port de communication
Procedure ComWrite (ComPort  :Byte; Value : String);
```

Exemple Portion de code

```
Uses Serial, Sysplc ;
Var
  Ok        : Boolean ;
  DoClose   : Boolean ;
  B         : Byte ;
  TS        : String ;

Begin
  Ok := OpenCom('1','9600','E','8','1') ; // Essaie d'ouvrir le port série
                                          // Parity = Paire, 8 dataBits & 1 stop Bit
  If GetComOpen(1)=False Then Message('Erreur Com1 pas ouvert');
  If DoClose And Ok  Then DisableCom(1) ; // *** Ferme le port série
End ;

Begin
// ** Test si il y a des caractères présent dans le buffer de Windows ****
 If V24DataOK (1) Then // *** Test de presence des cars sur com1
 Begin
    B  := GetByte(1) ; // *** récupère un car dans le buffer de Windows
    TS :=TS+Chr(B) ;   // *** Si est un car ASCII ajouter a la chaîne TS
    SendByte(2,B)      // *** Renvoie le byte sur Com2
    ComWrite (2,TS) ;  // ** Réémet la chaîne en totalité sur com2
 End ;
End ;

//  Envoie une série de byte dans le buffer d'émission
Procedure ComWriteBuf (ComPort:Byte; Var PointerTable; NB : Integer);
```

Exemple

```
 Var
    Tab : Array [0..100] Of Byte ;
 Begin
   ComWriteBuf  (3,Tab[0],101); // ** Envoie 101 bytes sur COM3
```

Onglet Times

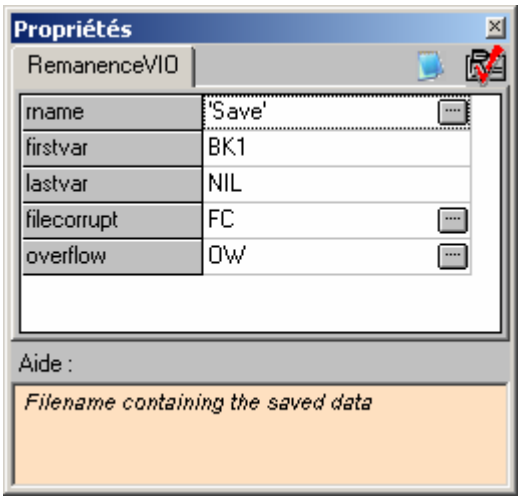
- Rémanence des données.
- Base de temps.
- Affichage Date et Heure
- Calendrier.
- Chronomètre
- Détection de changement d'états de variables
- Changement dynamique de langue.
- Temporisation d'un bit.

Composant RemanenceVIO

Le composant RemanenceVIO permet de sauver les données utiles du process et de les restaurer au démarrage de l'application. Les données sont compressées et stockées dans un fichier.

Exemple :
Supposons que les variables BK1, BK2, BK3, BK4 soient déclarées consécutivement dans les variables globales.

```
BK1   : Integer ;  // Variable sauvegardée
BK2   : Real ;     // Variable sauvegardée
BK3   : Boolean ;  // Variable sauvegardée
BK4   : String;    // Variable sauvegardée
RIEN  : Boolean;   // Talon
```



RName : Nom du fichier de sauvegarde.

FirstVar : Première variable du bloc de données.

LastVar : Dernière variable + 1 du bloc de données.

FileCorrupt : Indique une erreur dans le fichier à la restauration des données. La variable doit être remise à False par programme.

OverFlow : Indique un dépassement de la taille maximum du bloc de données compressées. La variable doit être remise à False par programme.

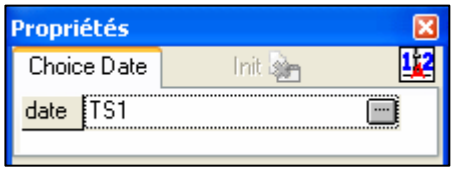
Le composant Remanence sauvegarde BK1 à BK4 dans un fichier nommé 'Save'.
La variable RIEN n'est pas sauvegardée.

La sauvegarde s"effectue de la variable **'FirstVar' à la variable 'LastVar'-1**.

Attention :
Le composant Remanence doit être placé sur le premier synoptique affiché au lancement de votre application afin d'être constamment exécuté (voir les fonctions Display et DisplayModal). Si vous intercalez des variables dans un block déjà sauvegardé, ces variables supplémentaires prendrons les valeurs des anciennes variables présentent à cet endroit. Il est donc impératif d'insérer les nouvelles variables juste avant le 'Talon'.

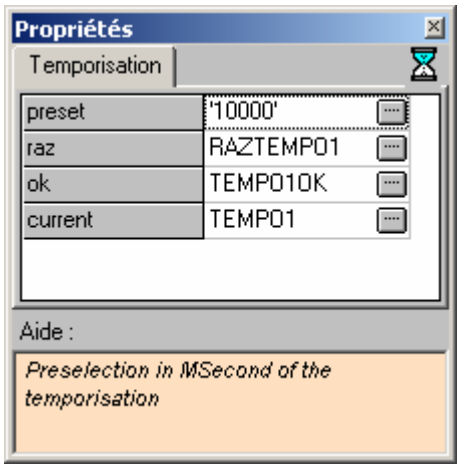
Les composants **Display time** et **Display Date** affichent respectivement l'heure et la date courante.
Ces 2 composants n'ont aucun paramètres à régler.

Le composant **Choix Date** permet de sélectionner une date à l'aide du calendrier de Windows.
Ce composant nécessite une chaîne de caractères recevant la date sélectionnée.



Composant Temporisation

Ce composant permet la mise en place de temporisations dans votre application. L'unité de temps utilisée est la milliseconde.



Preset : Présélection de la valeur de temporisation en milliseconde.

Raz : Remise à zéro de la temporisation.

Ok : Etat de la temporisation (encours ou terminée).

Current : Valeur courante de la temporisation. (Type Integer)

Attention :
Le composant Temporisation n'est actif que si le synoptique qui le contient est affiché à l'écran.

Une procédure Temporisation identique au composant Temporisation est disponible dans la librairie Temporisation. Elle peut-être utilisée dans un synoptique ou dans un programme cyclique.

Procedure Temporisation *(Preset: Integer; Raz: Boolean;Var ok: Boolean; Var Current: Integer);*

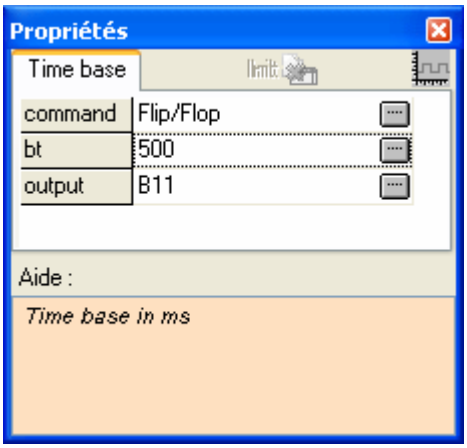
Exemple d'utilisation de la procédure Temporisation dans un synoptique :

```
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;
```

Composant TimeBase

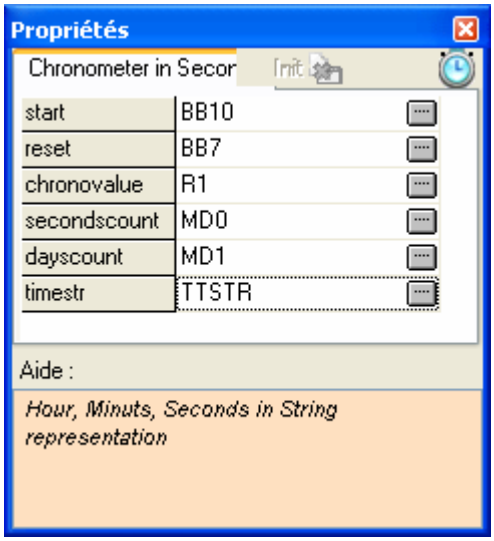
Permet de créer une base de temps qui permet d'inverser cycliquement un bit (Flip/Flop) le mettre à True (Set) le mettre à False (Reset) ou d'effectuer une impulsion sur un tour de cycle (Pulse).

Command Flip/Flop, Set, Reset ou Pulse.
Bt Constante ou variable exprimée en ms.
Output Bit à affecter au bout de la temporisation demandée.



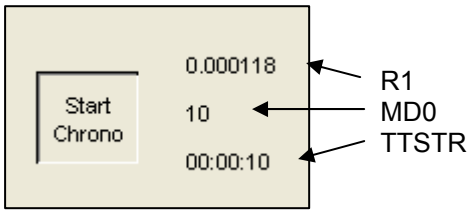
Composant Chrono

Permet de déclencher un chronomètre à l'aide d'un booléen.



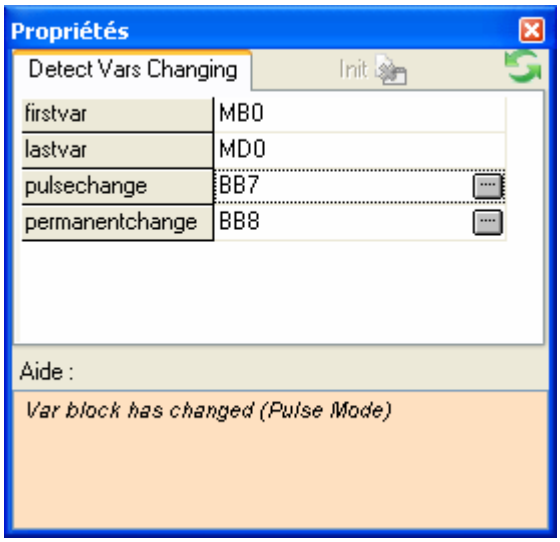
Start : Booléen déclenchant et arrêtant le chronomètre.
Reset : Booléen permettant de remettre à zéro le chrono.
Chronovalue : Réel représentant la durée sous forme d'une valeur numérique compatible avec les fonctions de gestion de la date et L'heure (DateTimetoStr..).
SecondsCount : Integer contenant les secondes écoulées.
DaysCount : Integer contenant les jours écoulés
TimeStr : String contenant les heures, minutes et les secondes écoulées

Exemple :



Composant Détection de variables changeantes

Permet de détecter qu'au moins une variable globale à changée d'état.
Cette ou ces variables sont contenues dans une série de variables consécutives déclarées dans le composant par la première variable et la dernière variable (non incluse).



Firstvar : Nom de la première variable dans le groupe à surveiller.
Lastvar : Dernière variable **NON INCLUSE** servant de talon Cette variable est celle directement déclarée après la dernière variable scrutée.
Pulsechange : Booléen fugitif passant à true quand une des variables surveillée change.Pulse sur 1 tour de prog.
PermanentChange : Booléen passant à True quand une des variables surveillée change. Ce booléen reste à True et le programme utilisateur doit le remettre à False.

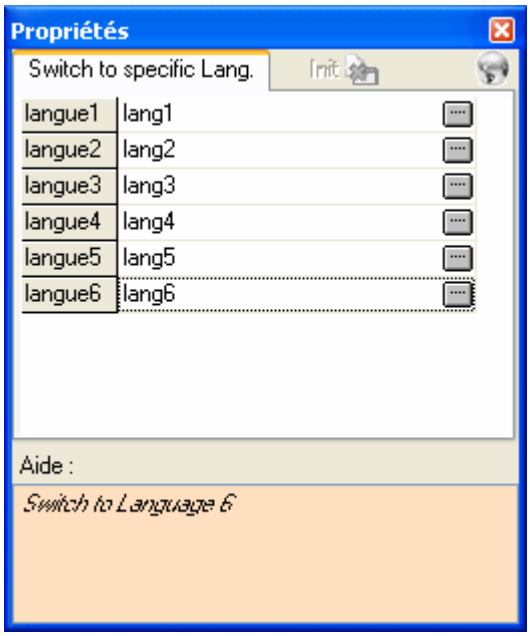
Exemple de variable globales :

```
{******* 8 Bits Words **************}
MB0,MB1,MB2,MB3,MB4,MB5,MB6,MB7,MB8,MB9          : Byte;  {** Etc.. **}
{******* 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;
```

Dans le paramétrage ci-dessus, seules les variables en Rouge sont surveillées. La variable MD0 ne l'est pas et sert exclusivement de talon.

Utilisez ce composant par exemple dans le paramètrage OpenModbus pour déclencher une trame de communication sur un changement d'état.
Utilisez plusieurs composants pour surveiller des blocs de variables différents.

Composant changment de langue



Langue1 : Boolean pour changer en francais.
Langue2 : Boolean pour changer en Anglais.
Langue3 : Boolean pour changer en italien.
Langue4 : Boolean pour changer en Allemand.
Langue5 : Boolean pour changer en Espagnol.
Langue6 : Boolean pour changer dans une autre langue que celles specifies ci-dessus.

Les caractères Unicodes sont utilisables.

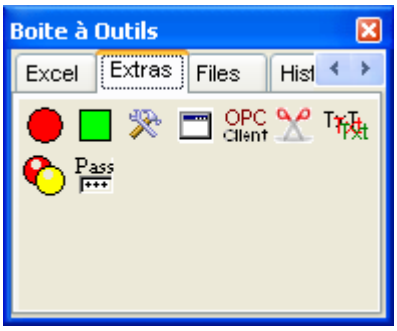
Onglet Trends

Cet onglet permet à l'aide de composants périphériques d'enrichir la courbe de base que vous trouverez dans l'onglet Avancé. Si une seule courbe est utilisée dans votre synoptique, tous les composants fonctionneront avec celle-ci.
Si plusieurs courbes doivent être utilisées, il faudra grouper les composants avec la courbe qui est associée.

Voir chapitre Création d'une page de visualisation de courbes

Onglet Extra

Cet onglet regroupe différents composants qui n'ont pas forcément de relation entre eux.

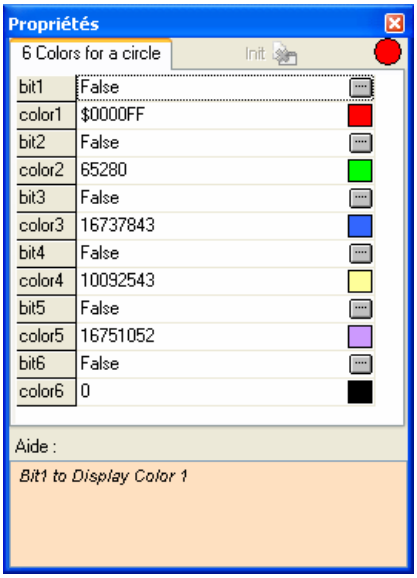


- Coloriage d'un cercle en fonction de 6 états
- Coloriage d'un rectangle en fonction de 6 états
- Affichage par glissement d'un synoptique
- Affichage et disparition d'un synoptique par un bit.
- Activation de la liaison OPC client paramétrée.
- Détection de non activité souris ou clavier.
- Coloriage d'un texte en fonction de 6 états
- Coloriage d'une LED en fonction de 6 états
- Champ de saisie avec mot de passe.

Coloriage d'un cercle en fonction de 6 états.

Coloriage d'un rectangle en fonction de 6 états.

Permet de tester 6 états (booléens) et de colorier le cercle en conséquence. Si aucun état est vrai le cercle n'est pas colorié.

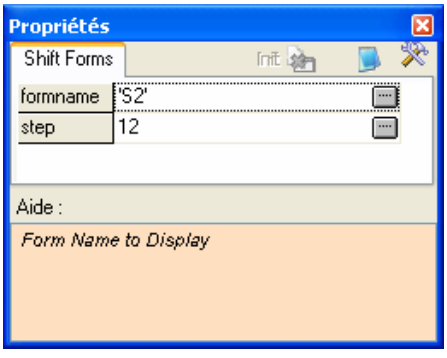


Bit1 à Bit6 : 6 Booléens (Bits) à tester. La priorité est effectuée de haut en bas (B1 à B6).

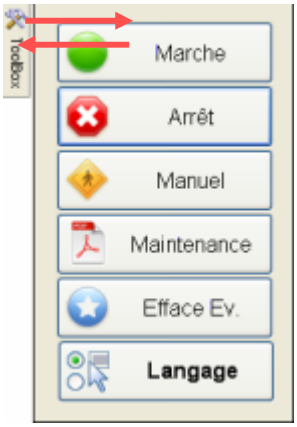
Color1 à color 6 : Les 6 couleurs correspondantes aux 6 bits.

Affichage et disparition par glissement d'un synoptique.

Ce composant permet d'afficher par glissement de la gauche vers la droite un synoptique. La disparition est effectuée si le synoptique perd le focus.



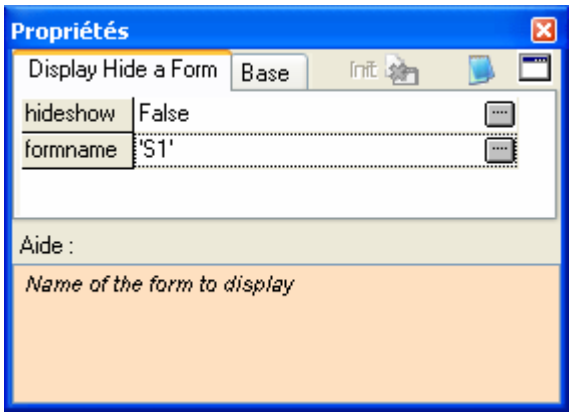
FormName : Nom du synoptique à faire apparaître. Le synoptique doit être inclus dans la liste des synoptiques de l'application.
Step : Vitesse d'apparition. Plus ce chiffre est élevé plus le synoptique glissera rapidement.



Ici le synoptique S2 apparaît en glissant de la gauche vers la droite quand l'opérateur passe la souris sur le Bitmap ToolBox. Le synoptique disparaît en glissant vers la gauche lorsqu'il perd le focus

Affichage et disparition d'un synoptique à l'aide d'un bit.

Ce composant permet d'afficher ou de cacher un synoptique en fonction de l'état d'un bit. Si le bit est à true alors le synoptique spécifié en affiché sinon le synoptique est caché.



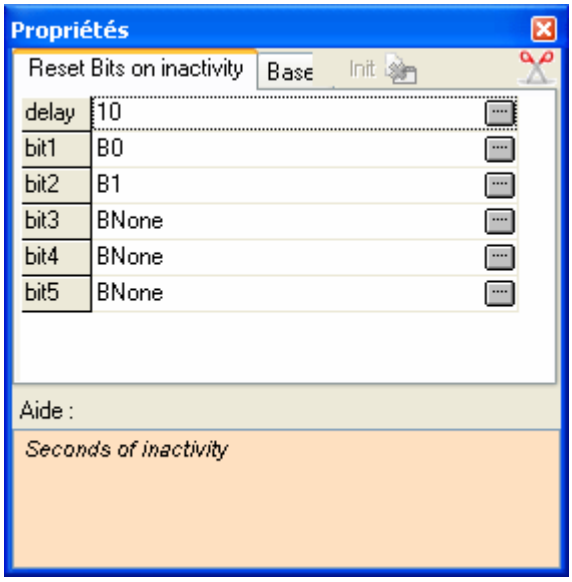
Hideshow : Bit pour affichage ou disparition du synoptique spécifié dans formname.
FormName: Nom du synoptique à afficher ou cacher ou variable chaine de caractères contenant le nom du synoptique.
Note :
Si le nom du synoptique est spécifié, celui-ci est encadré de 2 guillemets (Constante chaine de caractères).

Activation laison OPC Client.

Pour que la lecture et l'écriture des variables OPC du serveur soit effectuée, vous devez placer le composant OPC Client sur le synoptique principal afin que celui-ci soit exécuté cycliquement. Ce composant permet aussi à l'aide de sa propriété de stopper ou mettre en service le lien OPC.
Voir chapitre OPC client

Détection d'inactivité de la souris et du clavier.

Ce composant permet de remettre à 0 (false) 5 bits en cas d'inactivité de la souris ou du clavier.
Peut servir pour remise à zéro des autorisations ou mot de passe.



Delay : Temps d'inactivité (en secondes) avant de remettre à zéro les bits Bit1 à Bit5 spécifié.
Si un ou plusieurs paramètres bits ne sont pas utilisés, spécifiez BNone ou dupliquez les noms de variables déjà spécifiées.

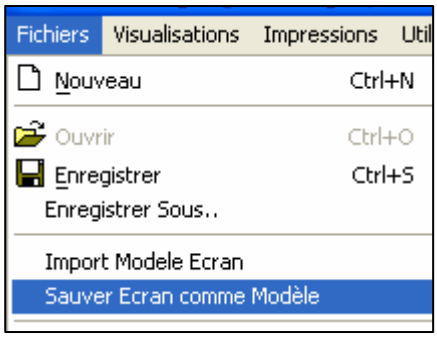
Bit1 à Bit5 : Bits remis à zéro en cas d'inactivité du clavier et de la souris.

Coloriage d'un texte en fonction de 6 états.
Coloriage d'une Led en fonction de 6 états.

Le texte et la led peuvent changer de couleur en fonction de 6 booleéns. Si aucun des booléens n'est à True, la led devient transparente et le texte retourne dans la couleur de tracé défini sous l'éditeur. Le texte a un attribut supplémentaire que n'a pas la Led, il peut être aussi clignotant.

Import Export d'écran type

Il est possible de sauvegarder des synoptiques en tant que modèles. Cette sauvegarde est effectuée dans le sous-répertoire Modèles du répertoire d'installation.



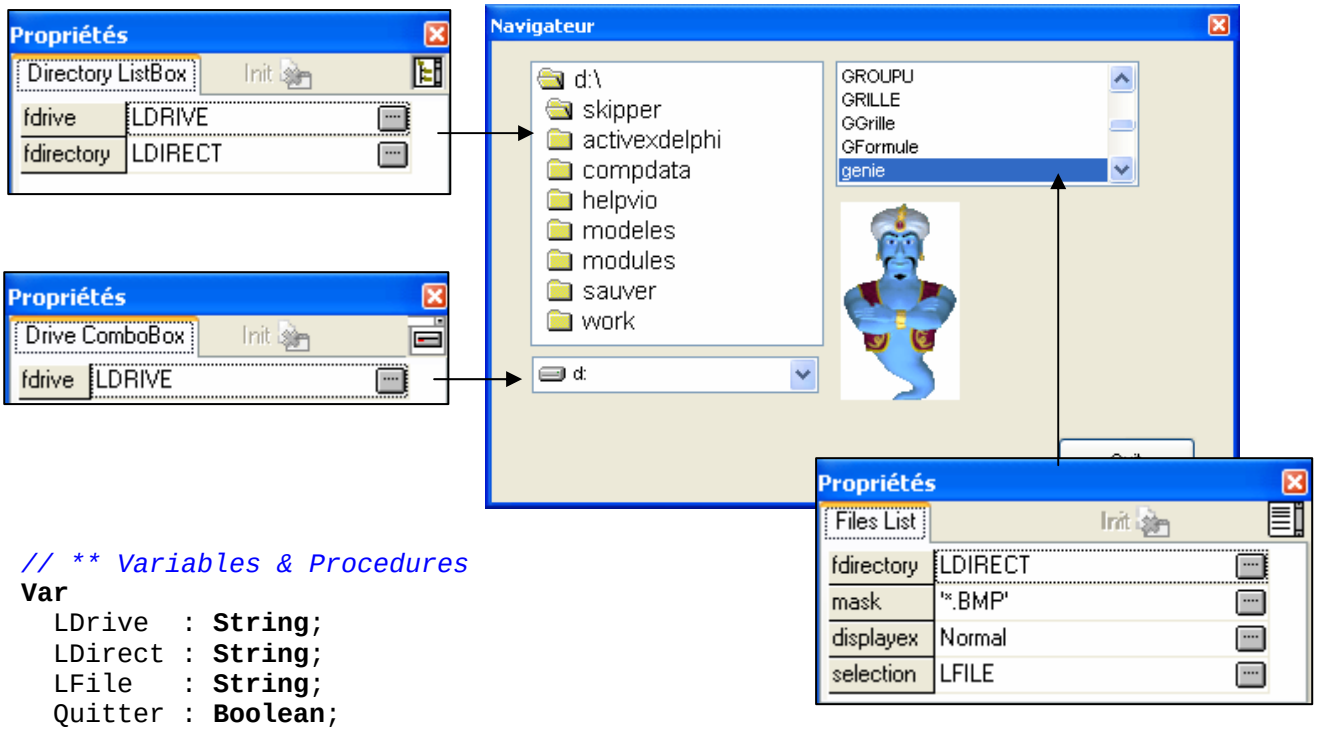
Les 2 commandes du menu Editer *Import Modèle Ecran* et *sauver Ecran comme Modèle*, permet de les sauvegarder et de les recharger dans n'importe quel projet.

Ces écrans sont sauvegardés avec leurs programmes.

Il est donc facile de créer des écrans types (synoptiques) de les tester puis de les sauvegarder comme modèles pour être utilisés comme base dans d'autres projets.

Bien sur le code source et le graphique est accessible et modifiable comme tout synoptique.
La version 7 de Visual I/O permet d'utiliser les variables déclarées dans le synoptique dans les experts des figures. Cette fonctionnalité permet de créer des modèles d'écrans sans avoir besoin d'utiliser des variables globales.

Exemple Navigateur et visualisateur de bitmaps




```
// ** Variables & Procedures
Var
  LDrive  : String;
  LDirect : String;
  LFile   : String;
  Quitter : Boolean;
// *** XPBUTTON1 ******************
SObject Procedure XPBUTTON1;
Begin
  If button then CloseAppli;
End;
```

☞ **Note** : Aucune Variable globale utilisée donc un écran complètement relogeable dans les applis.

Grouper

La commande Grouper (son inverse dégrouper) permet d'associer des figures graphiques entre elles.
Cette commande permet lorsqu'un des éléments graphiques sont sélectionnés sous l'éditeur, que l'ensemble de ses éléments soit aussi sélectionné.
Ceci à pour effet de maintenir les espacements graphiques entre eux quoi qu'il arrive.

La seconde fonction de grouper est de pouvoir associer dans un programme une commande à l'ensemble du groupe.

Un groupe de figures à un premier et un dernier élément. Lorsque vous grouper des figures le premier du groupe est le premier tracé dans ce groupe. le dernier suit le même raisonnement.
Pour modifier l'ordre des éléments utilisez la commande 'PasserDevant' ou 'PasserDerrière'.

Les éléments graphiques dans Visual I/O ont tous la même structure interne. Il est donc possible de leur appliquer les mêmes instructions.

Prenons le cas d'un groupe contenant un cercle, un rectangle et un bouton.

Next_Group

Appliquons sur le premier élément de ce groupe (le cercle) le programme suivant :

```
Begin
  Repeat
    SetFillColor(255);   {*** colorie l'intérieur des figures en Rouge **}
  Until Next_Group=False;
End;
```

SetFillColor(255); Colorie l'intérieur d'une figure avec la couleur spécifiée (ici Rouge)

La première fois, SetFillColor est appliquée au cercle.
Next_Group Passe sur la figure suivante dans le groupe (équivaux à faire un Commut_fig sur la figure suivante).
Next_Group renvoi False quand il est arrivé à la dernière figure.

Le processus de coloriage (SetFillColor) est appliqué alors à la nouvelle figure et ainsi de suite, jusqu'à atteindre la dernière.

Cas ou l'on ne soit pas sur la première figure d'un groupe.

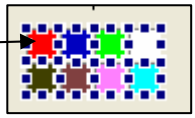
```
Begin
    Repeat Until Previous_Group=False;  {*** Remonte jusqu'à la première figure***}
    Repeat
       SetFillColor(255);  {** colorie l'intérieur des figures en Rouge **}
     Until Next_Group=False;
End;
```

Previous_Group

Previous_Group remonte les figures dans le groupe jusqu'à la première. Ce qui permet d'être sur de démarrer véritablement de la première figure pour faire le traitement

```
 Repeat Until Previous_Group=False;  {*** Remonte jusqu'à la première figure***}
```

Exemple de sélection d'une couleur par drag & drop. La première figure (carré ducoin haut gauche) contient le programme suivant. Les autres carrés sont simplement groupés.



```
{*** Shade1 **********************}
SObject Procedure Shade1;
Begin
  {*** Scan le Groupe ****}
  Repeat
    Drag(IntToStr(GetFillColor)); //** Intercepte le click souris
  Until Next_group=False;
End;
```

Si un click souris est effectué sur un des 8 carrés le mécanisme de Drag & Drop est déclenché.

Groupe de figures dans un composant.

Next_Group_Compo
Previous_Group_Compo

Les instructions Next_Group_Compo et Previous_Group_Compo fonctionnent de la même manière que Next_Group et previous_group mais parcourent que les figures groupées d'un composant.

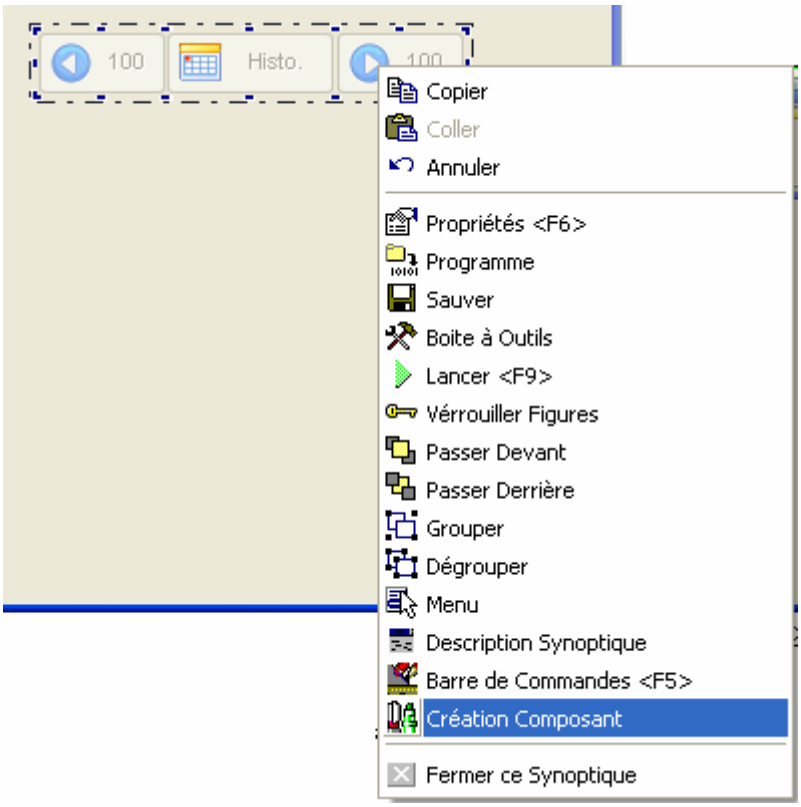
Exemple

```
Repeat
Until Next_Group_Compo=False;    //*** Se positionne au Premier Bouton du composant
If Button Then
Begin
  …
 Exit;
End;
Previous_group_compo;   //** Va au deuxième Bouton du composant
If Button Then
Begin
  …
End;
```

Le composant ShHisto de l'onglet Trends.

Le groupe de figures d'un composant est fait automatiquement en sélectionnant toutes les figures du composant puis clic droit de la souris et sélection de "Création Composant" du menu flottant.

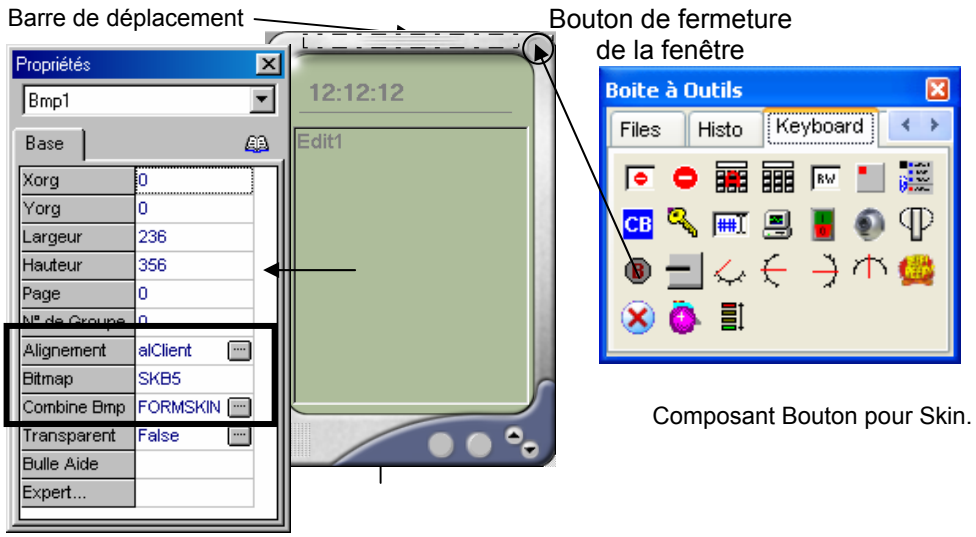
Le composant créée est constitué de toutes les figures sélectionnées



Facades (Skins)

Il est possible dans Visual I/O d'obtenir un synoptique sans barre de titre ni bordure. De designer votre propre contour de fenêtre sans vouloir tenir compte de la présentation de fenêtres Windows.
Appelées Façades ou Skins, la fenêtre d'un synoptique peut revêtir tous les aspects possibles. Une démonstration est visible dans Exemple PROGRAMMATION du CD d'installation générale de Visual I/O.

Exemple



Composant Bouton pour Skin.

Mode d'emploi :
Placez votre Bitmap Skin en premier sur un synoptique Vierge.
Sélectionnez le puis frappez F6 (Appel boîte de propriétés).
Fixez la propriété **Alignement** à la valeur **alClient**.
Fixez la propriété **Acombine Bmp** à la valeur **FORMSKIN**.
Le bitmap devient alors le fond de plan de synoptique.
☞ **Note** : pour revenir à l'aspect normal Windows, Fixez Combine Bmp à la valeur Normal.

La Barre de déplacement

Sélectionnez un cadre Image dans la boite à outils définissant le contour de la barre de déplacement du Skin.

Placez ce code derrière cette image :

```
SObject Procedure Bmp2;
Begin
  If TopLBDown then Perform(WM_SYSCommand, $f019,0);
End;
```

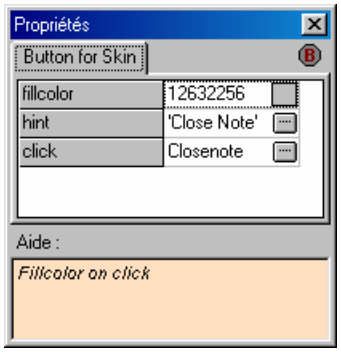
Placez aussi la directive **Uses Windows** au début de programme.

```
// ** Variables & Procedures *
Uses Windows;
```

Cette Vpu contient en autre la procédure Perform. En run l'opérateur pourra utiliser cette zone à la souris pour déplacer sur l'écran le synoptique Skin à la place de la barre de titre habituelle.

Composant *Button For Skin* de l'onglet keyboard

Ce composant peut être placé pour n'importe quelle action de l'opérateur. *Une fermeture du synoptique par Exemple* .En double cliquant sur ce bouton placé sur le Skin, une boite de propriété apparaît :



FillColor : Est la couleur de remplissage du cercle lorsque l'opérateur clic sur ce bouton.
Hint : Est le message d'aide qui apparaît lorsque le curseur de la souris passe sur ce bouton.
Click : Est la variable qui passe à True au relâchement de la souris sur ce bouton.
Le code associé pour une fermeture de synoptique Skin pourrait être :

```
If CloseNote Then
  Begin
    CloseAppli;   // ** Femeture du Synoptique *
    CloseNote:=False;  // ** Reset de la variable *
  End;
```

Traduction des synoptiques

Visual I/O permet les applications multi Langues. Une application constituée de synoptiques contenants des textes statiques, Cases à cocher, Menus etc.., peut être traduite grâce au mécanisme du dictionnaire.
Le dictionnaire est un fichier texte composé de colonnes. Chaque colonne est séparée par un séparateur qui est le caractère tabulation (#9).
La première colonne est le français ensuite Anglais, Italien, Allemand, espagnol et autre. La colonne autre est utilisée dans le cas où l'on n'utilise pas une des langues prédéfinies.
Il est très facile alors de faire correspondre les mots dans les différentes langues associés aux colonnes correspondantes. Utilisez le dictionnaire à colonne dans l'éditeur pour modifier, visualiser le fichier Dico.lng.
Ex : 1 Ligne du fichier Dico.lng.

Confirmez #**9** Confirm #**9** Confermare #**9** bestätigen #**9** Confirme #**9** подтвердите
Français Anglais Italien Allemand Espagnol Russe (ou autre langue)

Tous les synoptiques essaient de ce traduire automatiquement par rapport au fichier **Dico.lng** se trouvant dans le répertoire de votre application. Si ce fichier n'est pas créé et que la langue est différente de Français, l'éditeur de Visual I/O crée le fichier et ressort tous les textes des synoptiques de l'application.
Ce fichier est compatible avec excel. La langue est fixée au démarrage de l'application basée sur la langue utilisée par Windows. Pour changer de langue cliquez sur :
Dans paramètres et panneau de configuration du menu Démarrer de Windows.

La procédure SetLanguage

Procedure **SetLanguage**(NumLang : Integer);
Cette procédure change dynamiquement de langue. Tous les textes de votre application sont retraduits et réaffichés dans la langue spécifiée. Numlang varie de 1 à 6 (Français à Autre).

La fonction Translate

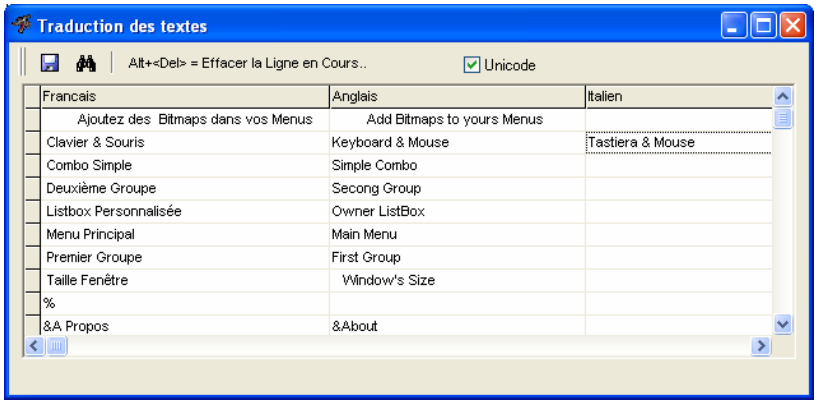
Function **Translate**(N : String) : String;
Cette fonction renvoi le texte français traduit ceci par rapport au dictionnaire Dico.lng.

La variable DoTranslate

DoTranslate : Boolean=True;
Variable permettant d'interdire la traduction automatique. (False interdit, True (par défaut) autorisée).
Si une de ces fonctions de doit pas être traduits, insérer la commande DoTranslate:=false avant.
Exemple :

```
DoTranslate :=False;
  WinMess(Translate('Ouvrir')+NomFichier) ; // *** Ici chaîne composée
DoTranslate :=True;
```

Appel et modification du dictionnaire



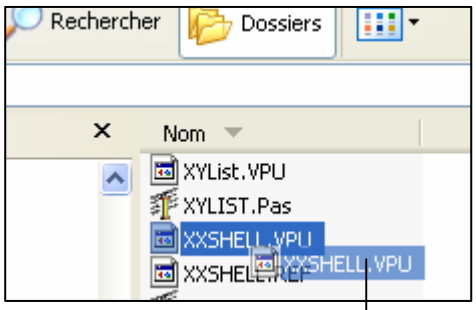
Le dictionnaire est une grille à plusieurs colonnes permettant de rentrer les différents textes utilisés dans l'application dans les différentes langues. Les caractères Unicode peuvent être utilisés, autorisant ainsi l'affichage de caractères cyrilliques arabes chinois etc.. Si une langue n'a pas de colonne, il faudra rentrer les textes dans la colonne autre. Si en exploitation la langue n'est pas trouvée le texte Anglais est choisi.

Le Drag & Drop ou Glisser Déposer

Action par laquelle l'utilisateur sélectionne avec le pointeur de la souris un objet à l'écran, le déplace jusqu'à une autre position en maintenant le bouton de la souris appuyé, puis le lâche.
Une manière très sympa de créer des interfaces graphiques *friendly interface* est de permettre de faire du drag & drop de tout ce que l'on veux dans, depuis et à l'intérieur même de l'application.

L'origine : DRAG
La ou les destinations : DROP et GetDrop.

De l'explorateur



Venant de listbox1

Drag(FDirectory);

Permet définir un point de démarrage possible d'un drag & drop.
Si l'utilisateur démarre le mécanisme de listbox1, la chaine FDirectory est mémorisée afin d'être transporté vers un autre élément de destination.

DragAcceptFiles*(GetHWnd,True);* cette fonction derrière un control autorise le Drag & drop venant de l'explorateur de windows.

Drag(S : String) ;
Permet de stocker la chaîne de caractères servant de transfert lors d'une opération de drag and Drop.
La chaîne S est chargée au moment du click gauche de la souris dans une chaîne de caractères interne servant de transport vers un élément de réception (Drop). Le curseur est changé en curseur interdit.
Si l'opérateur relâche de suite le bouton de la souris, le curseur redevient normal et la chaîne interne est remise à zéro. La chaîne interne est déchargée dans un élément contenant l'instruction Drop. Cet élément de destination est signalé par le changement de curseur en flèche.

Drop : Indique que la figure est un élément de destination du Drag and Drop. Quand le curseur de la souris survole cet élément la forme du curseur change. Drop renvoie true si la souris est relachée sur l'élément.

GetDrop : Renvoie le contenu de la chaine chargée par DRAG.
DragFrom : Renvoie le nom de la figure de départ du Drag & Drop.

```
{*** ListBox2 ************************************}
SObject Procedure ListBox2;
Var
  TS:String;
Begin
   If Start_visu Then
     DragAcceptFiles(GetHWnd,True); //** Accept files from windows' explorer
  If Drop Then
   Begin
     {*** Recept Selective Drop ****}
     If DragFrom = ('LISTBOX1') Then Addstring(GetDrop)
                                Else SetFillColor(StrToInt(GetDrop));
   End;
End;
```

Exploiter un ActiveX

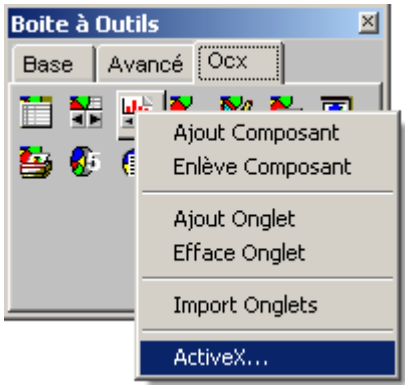
"Arme redoutable pour tout développeur qui se respecte" !

Technologie de Microsoft reposant sur OLE (Object Linking and Embedding) et COM (Component Object Model).
Les contrôles ActiveX sont en fait des composants COM proposant des particularités supplémentaires.
En effet, pour utiliser un composant COM vous devez "mettre les mains à la pâte" et donc coder.
Un contrôle ActiveX facilite sérieusement son utilisation en le rendant visuellement manipulable par Visual I/O.
En Visual I/O, vous pouvez poser tout contrôle ActiveX sur votre synoptique.
Vous changez ses propriétés via la boite de propriétés et vous codez facilement grâce aux procédures et méthodes extraites de cet activeX.
Mais attention ! Être visuellement manipulable dans Visual I/O ne veux pas forcément dire que le contrôle sera visible à l'exécution. Il existe effectivement des contrôles non visible à l'exécution : le contrôle Timer par exemple.

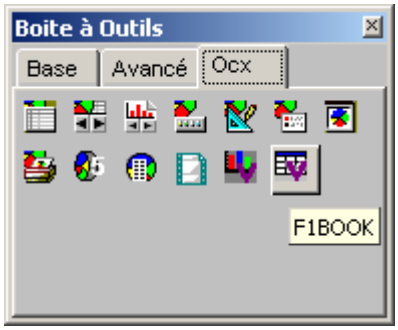
Généralement, le code des contrôles ActiveX se trouve dans un fichier d'extension ".OCX". Il s'agit en fait d'une DLL classique (Dynamic Linking Library).
C'est pour cette raison que l'on appelle familièrement les Activex les OCX.

Scanner les ACTIVEX installés dans votre PC.

Appelez la boite à outils, créez éventuellement un onglet supplémentaire (par exemple OCX) puis faire clic droit avec la souris



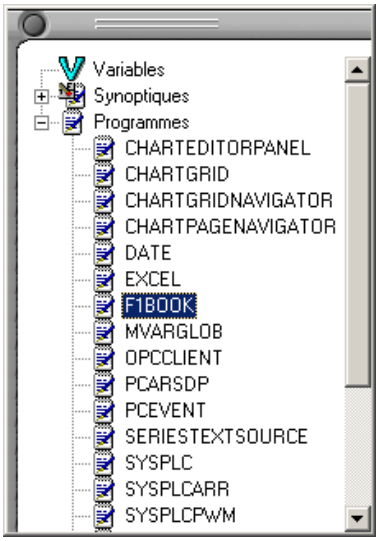
La liste des OCX **installés** dans votre PC est affichée. Choisissez simplement en cochant la case à cocher à gauche de l'ocx choisi.

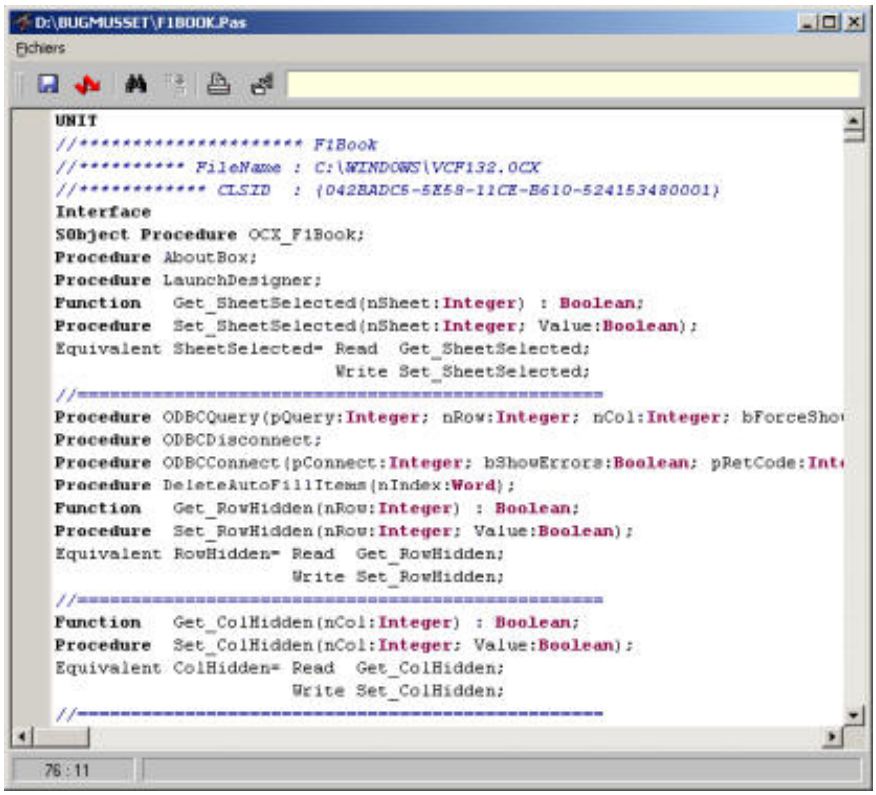


◄ Les 2 ActiveX sont alors placés dans la boite à outils.
Vous pouvez maintenant les utiliser comme un composant Visual I/O classique.

2 librairies VPU sont alors générées automatiquement donnant accès à toutes les procédures et les fonctions implémentées dans l'ocx.
Si l'OCX s'appelle **F1BOOK** le composant Visual I/O s'appelle **F1BOOK.VPU**

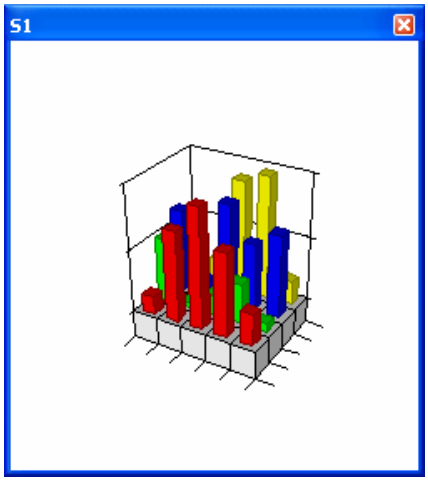
Avec l'Activex nous rentrons directement dans la programmation en langage évolué de Visual I/O.
Consultez les rubriques concernant la programmation, les librairies et le langage avant de plonger.

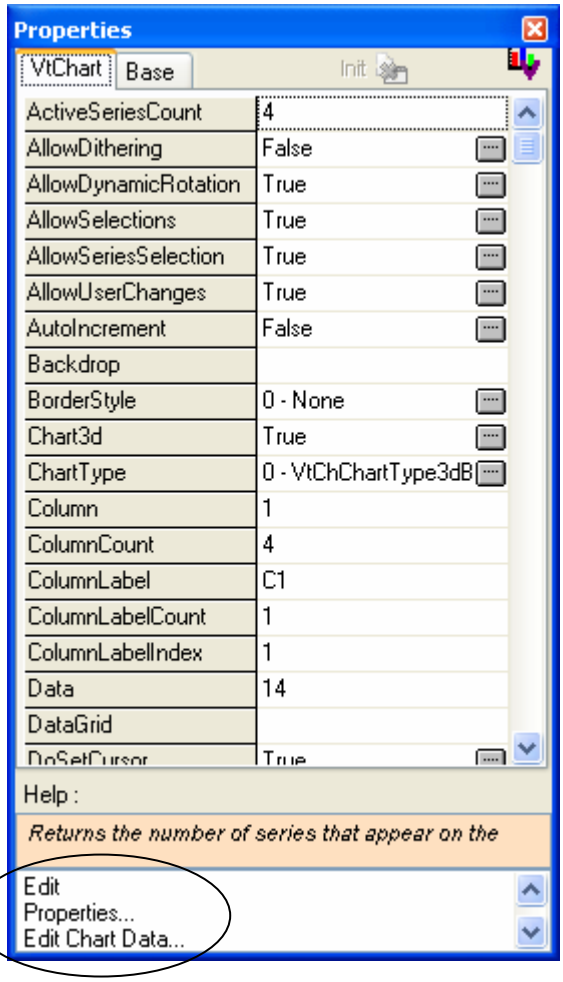




Placer l'ActiveX VTCHART

L'ActiveX affiche des réglages par défaut que vous pouvez modifier à l'aide de la boite de propriétés (appel par un double clic).





<F1> permet d'appeler l'aide fournie avec l'OCX.
Aide permettant généralement de documenter les procédures internes de l'OCX.

Derrière cet ActiveX dans la rubrique programme vous allez aussi pouvoir utiliser les procédures et méthodes qui
lui sont propres à cet ActiveX.

La liste du bas permet par double clic d'appeler les boites de dialogues internes à l'ActiveX

Programmer un ActiveX

Après avoir placé votre OCX sur le synoptique, vous allez pouvoir utiliser ses procédures et ses fonctions afin de le contrôler à votre guise. Dans le fichier VPU généré automatiquement lors de l'ajout de l'ActiveX dans la boite à outils, vous trouverez plusieurs types d'infos.

Les constantes

Qui seront plus explicites que de simples valeurs. Ces constantes sont aussi généralement en références dans l'aide de l'ActiveX.

Exemple :

```
// *** FontStyleConstants
VtFontStyleBold = 1;
VtFontStyleItalic = 2;
VtFontStyleOutline = 4;
// *** FontEffectsConstants
VtFontEffectStrikeThrough = 256;
VtFontEffectUnderline = 512;
// *** FrameStyleConstants
VtFrameStyleNull = 0;
```

Les propriétés

Qui sont utilisées dans les programmes comme des variables mais en fait qui font référence à 2 procédures (une en lecture et l'autre en écriture).

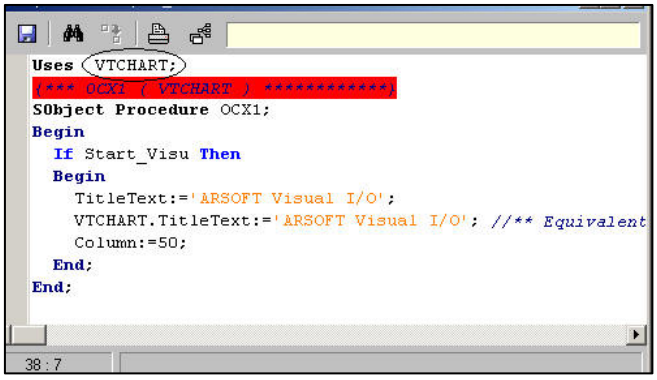
Exemple :

```
// ================================================
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;

Quand le programme lit   la fonction GET_xxx est appelée
Quand le programme écrit la procédure SET_xxx est appelée
```

En programme vous écrirez :

```
CPT:= Colum;  // *** Renvoi le nombre de colonnes appel auto de Get_Column
Colum:=12;    // **** Fixe le nombre de colonnes  appel auto de Set_Column
```

Exemple de programme sur VTChar



Ici ***TitleText*** permet de fixer le titre du graph.
La directive Uses VTCHART est nécessaire pour que le compilateur trouve les procédures, variables fonctions utilisées.

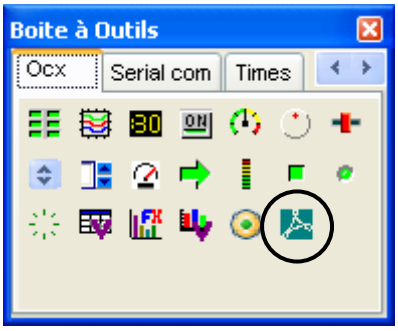
Ecriture VTCHART.TitleText permet de spécifier la vpu qui contient la procédure TitleText ceci dans le cas
ou on utiliserai 2 ActiveX distinct contenant un nom de procédure similaire (ex : ABOUT).

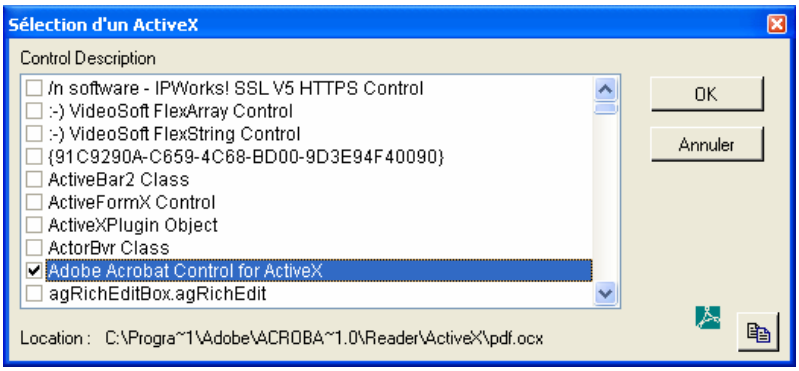
Note:
Si le synoptique ne contient qu'un ActiveX alors les méthodes (procédures et fonctions) de l'ActiveX peuvent être appelés de n'importe ou. sinon un **Commut_fig**('OCX...') doit être effectué pour ce placer derrière l'OCX voulu

Exemple:

```
Begin
  Commut_Fig('OCX1');
  About;
End;
```

L'activex d'adobe acrobat





Sélectionnez l'activeX *d'adobe Acrobat Control* pour l'utiliser dans votre application.

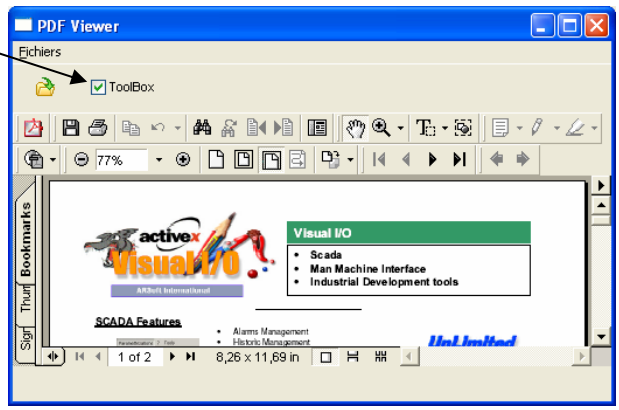
Le contrôle activeX vient se placer dans la boite à outils dans l'onglet Choisi avant le scan des ActiveX (voir chapitre précédent). Avec ce composant Visual I/O génère une VPU d'interface pour Acrobat nommée PDF.PAS et PDF.VPU ou l'on trouve une ces fonctions suivantes.
A ce stade il est nécessaire de se rapprocher d'Adobe pour avoir le détail des ces fonctions.

```
Procedure AboutBox; //*** Ranvoi la boite apropos
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;
```

Quelques noms de fonctions sont intuitives comme Aboutbox qui affiche une boite A propos, printall qui imprime le contenu du fichier PDF ou loadFile qui charge et visualise un fichier PDF.
Grace au mécanisme d'équivalence de Visual I/O, La pseudo variable **SRC** définie permet de lire ou de changer le nom du fichier PDF à visualiser.
Pour changer le nom du fichier à visualiser dans l'ocx Acrobat 2 possibilités sont offertes :
Uses PDF;

```
If OpenPDF Then PDF.Loadfile(PDFName);  //** Appel de la procedure LoadFile
Ou
If OpenPDF Then PDF.SRC:=PDFName; //** Affectation de la pseudo Variable SRC
```

Ici le boolean utilisée **ToolBox** est l'état de la case à cocher.




```
{*** OCX1 ( PDF ) ************}
SObject Procedure OCX1;
Begin
  If OpenPDF Then
  Begin
     PDF.Loadfile(PDFName);
     PDF.setShowScrollbars(ToolBox);
     PDF.setShowToolbar(ToolBox);
  End;
End;
```

Les Variants

Une variable de type Variant peut prendre n'importe quel type. Cela constitue un moyen commode de manipuler une variable sans connaître son type au moment de la compilation.
ATTENTION, cette commodité se paie au détriment d'un code plus volumineux et plus lent. Toutes les opérations sont autorisées par le compilateur, ce qui fait que la majorité des erreurs apparaissent en exécution. Il faut donc les utiliser avec parcimonie.
Les opérations où les variants sont utilisés sont plus lentes que celles portant sur des types associés statiquement. Enfin, les opérations illégales portant sur des variants provoquent fréquemment des erreurs d'exécution, alors que les mêmes erreurs avec des variables normales sont détectées à la compilation.

Visual I/O permet de travailler sur des variants grâce à la librairie **XVariants.pas**. Le variant se détermine ainsi comme un record contenant 16 octets. Dans la Vpu XVariants.pas il est défini comme ceci :

```
Variant = Record
     VType     : Word;  //** Type of Variant
     Reserved1 : Word;
     Reserved2 : Word;
     Reserved3 : Word;
     Value     : Array [1..8] Of Byte; //** Pointeur, valeur selon type
  End;
```

Vtype peut prendre ces valeurs ou la combinaison des ces valeurs

```
  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;
```

Si VType est égal à **VarInteger** Alors Value[1] à Value[4] Contient l'integer etc..

Généralement il nous est retourné un pointeur sur variant puisque le retour de fonction ne peut renvoyer que des registres. Dans le cas d'un pointeur sur Variant il est nécessaire de récupérer la totalité du variant pointé dans un variant local. Utilisez la procédure **PVar2Var**.
Exemple :

```
Var
 P : Pointer;
 V : Variant;
Begin
  P:=ReturnVar.. //** Fonction X retournant un Pvariant
  PVar2Var(P,V); //** La procédure permet de copier dans un variant (V)
                 //** local pour traitement.
End;
```

Pour récupérer une valeur dans le variant utilisez les fonctions :

```
Function  VariantToBool   ( Var V : Variant) : Boolean;   //** Returns a Boolean from Variant
Function  VariantToInt    ( Var V  : Variant) : Integer;  //** Returns an Integer from Variant
Function  VariantToWord   ( Var V : Variant) : Word;      //** Returns a Word (Smallint) from Variant
Function  VariantToSingle ( Var V : Variant) : Single;    //** Returns a Single from Variant
Function  VariantToDouble ( Var V : Variant) : Double;    //** Returns aDouble from Variant
```

Pour affecter une valeur dans un variants utilisez les procédures :
Procedure **VariantFromBool** (Var V : Variant; Value : Boolean); *// Affecte un booléen à un variant*
Procedure **VariantFromInt** (Var V ; Value :Integer); *//** Affecte un integer à un variant*
Procedure **VariantFromWord** (Var V ; Value :Integer); *//** Affecte un Smallint à un variant*
Procedure **VariantFromSingle** (Var V ; Value : Single); *//** Affecte un Single à un variant*
Procedure **VariantFromDouble** (Var V ; Value : Double); *//** Affecte un Double à un variant*

Exemple :

```
Var
  I,J : Integer;
  V   : Variant;
Begin
   VariantClear(V); //** Raz du variant V
   I:=123;
   VariantFromInt(V,I); //** Affecte la valeur 123 à V
   J:=VariantToInt(V); //** J devient égal à 123
End;
```

Variant contenant un tableau

Le variant récupéré peut être un tableau (VarArray) dans ce cas il est nécessaire de connaître sa dimension son indice de démarrage et son indice final. Pour cela des fonctions sont disponibles dans la librairie XVariants.

```
//** Cette fonction vous indique si le variant contientt un tableau.
Function  VarIsArray (Var A: Variant) : Boolean;

//** Cette fonction renvoie la dimension du tableau dans le variant
Function VarArrayDimCount  ( Var A: Variant): Integer;

//** Cette fonction renvoie le premier indice du tableau  Ex : 1 Array [1..?]
Function VarArrayLowBound  ( Var A: Variant; Dim: Integer): Integer;

//** Cette fonction renvoie le dernier indice du tableau (Ex : 10   Array [?..10]
Function VarArrayHighBound ( Var A: Variant; Dim: Integer): Integer;

//** Cette fonction verrouille les données dans le tableau de variant et retourne un pointer sur les données.
Function VarArrayLock   ( Var A: Variant): Integer;

//** Cette fonction déverrouille les données verrouillées précédemment
Procedure VarArrayUnlock ( Var A: Variant);
```

Cas ou l'on connaît la taille et le type des données dans le tableau. Exemple un tableau de 10 integers dans le variants est à récupérer dans un tableau local (Tablo).

```
Var
  Tablo : Array [0..9] of Integer; //** reçoit les datas du variant
  P  : Integer;
  PC : Integer;
  V  : Variant;
Begin
   PVar2Var(P,V);
   PC:=VarArrayLock(V); //** Renvoie un Pointer sur les données
   Move(PC^,Tablo[0],Sizeof(Tablo)); //** effectue un Block move de données
   VarArrayunLock(V); //** Déverrouillage des données.
End;
```

Cas on ne connait pas la taille du tableau ou le tableau de variants est de type différents à chaque indice. Il est possible de récupérer un tableau dans un variant ou l'indice 1 est un integer, l'indice 2 un boolean l'indice 3 un double. Dans ce cas il faut récupérer indice par indice et traiter le variant récupéré en fonction de son type.

Exemple :

```
Var
 A,B    : Variant;
 P,I,J  : Integer;
 TB     : Boolean;
 index,LowVal,HighVal : Integer;
 S      : Single;
Begin
  if Button Then
  Begin
     P:=Func..; //** retour un pointeur sur variant
     PVar2Var(P,B); //** Stockage local du variant
     TB:=VarIsArray( B); //** test si c'est un tableau
     If Tb Then
     Begin
        Index   := VarArrayDimCount(B); //*** Dimension tableau
        lowVal  := VarArrayLowBound(B, index); //** Premier Indice
        highVal := VarArrayHighBound(B, index); //** Dernier Indice
        For J := lowVal to highVal Do //** Balaie le tableau
        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;
```

Retour de fonction d'un ActiveX le type PVariant

Certains ActiveX peuvent retourner un variant. Dans ce cas En Visual I/O la fonction retournera un pointeur sur un variant (variant non permanent). Il faut alors le transférer dans une variable locale Visual I/O.

Exemple :
Dans une VPU la fonction suivante
Function ReadBinary(maxBytes:Integer) : PVariant;

Exploitation dans un programme

```
Uses XVariants;
Var
  V : Variant;
  P : Integer;
Begin
    P:= ReadBinary(100); //** Récupère le Pointeur sur Variant
    PVar2Var(P,V);  //** Transfert le contenu du variant retourné dans
                    //   le variant local
End;
```

Programmation

Comme Il est souvent difficile de réaliser une application qu'à l'aide de boites de dialogue. Visual I/O permet la programmation en langage Pascal. Inspiré du pascal de Borland et tiré de produits comme Visual Pascal et ML-Visu autres outils de développement d'ARSOFT International, Visual I/O s'appuie sur le langage scientifique des années 70 et continue à promotionner celui-ci .

Visual I/O met à la disposition du programmeur un langage évolué ainsi que des centaines (pour ne pas dire des milliers) de procédures et de fonctions permettant de manipuler tous les objets graphiques se trouvant dans les synoptiques. Le PASCAL, créé par WIRTH au début des années 70, possède des instructions assez claires (si vous comprenez l'anglais), et favorise une approche méthodique et disciplinée (on dit "structurée").

Le PASCAL de Visual I/O est un langage compilé, c'est à dire qu'il faut :

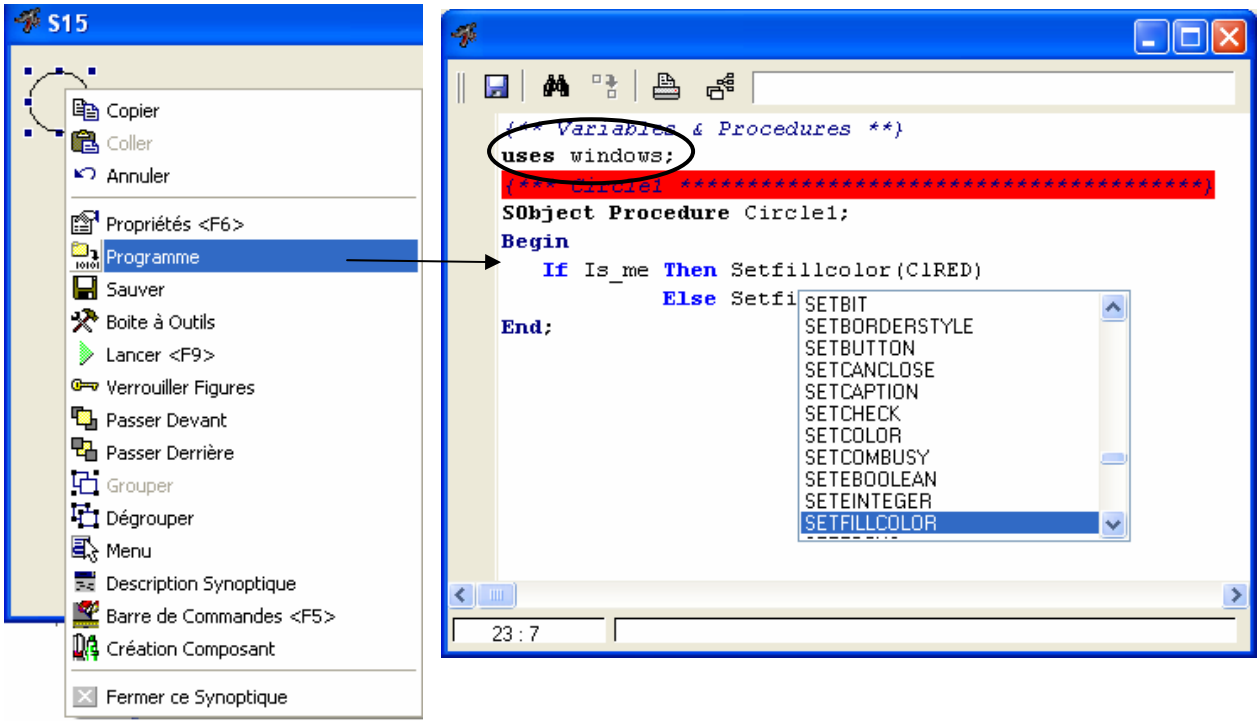
- Entrer un texte dans l'ordinateur (à l'aide d'un programme appelé EDITEUR),
- Le traduire en langage machine (c'est à dire en codes binaires compréhensibles par l'ordinateur) : c'est la compilation,
- Puis l'exécuter. Code englobé dans un seul fichier appelé exécutable (à l'extension .EXE)

Contrairement à un basic (ou VB) interprété, l'exécution sera beaucoup plus rapide puisqu'il n'y a plus de traduction à effectuer. De plus contrairement à d'autres produits l'exécutable créé par VisualI/O ne nécessite aucune grosses DLLs à installer. Vous pouvez diffuser votre application sur une simple disquette.

Bien que le langage soit normalisé, un certain nombre de points ont été améliorés pour les applications de supervision .Comme le test d'un bit dans un mot (W0.0), les Micro objets,les paramètres optionnels etc.. Ces indications ne seront pas toutes données ici.

Création d'un programme derrière un objet graphique dans un synoptique

Sélectionnez une figure quelconque (exemple un cercle) puis cliquez avec le bouton droit de la souris pour faire apparaître le menu flottant.



Derrière chaque élément graphique, vous pouvez le manipuler à l'aide de procédures et de fonctions.
Achèvement du code : Une aide intuitive permettant de choisir dans une liste de procédures et de fonctions par rapport au texte déjà frappé.
Le code ici:

```
If IS_ME                    // Renvoi true si la souris est dans le cercle
Then SetFillColor(CLRED)    // Colorie l'intérieur du cercle en Rouge.
```

CLRED est une constante se trouvant dans la Librairie **WINDOWS.VPU**. c'est pour cette raison que la déclaration
Uses Windows; existe en début de programme.

Les commentaires

Les commentaires sont des textes ignorés par le compilateur. Ils permettent de renseigner le programmeur sur des points particuliers qui sont ainsi mis en évidence.

Les commentaires sont mis entre accolades // et *{Ceci est un commentaire*
ou démarre par une double barre et toute la ligne est un commentaire *// **** ceci est un commentaire*

Les identificateurs

Pour écrire un programme, on utilise des identificateurs qui symbolisent certaines parties de sa structure. Un identificateur Pascal doit avoir un maximum de 63 caractères et obligatoirement commencer par une lettre de l'alphabet. Ces caractères sont les lettres non accentuées, les chiffres et le caractère de soulignement. Aucune distinction n'est faite entre les majuscules et les minuscules a et A sont donc identiques.

Liste des caractères acceptés :

a à z	A à Z
0 à 9	et _

Voici quelques bons exemples :

AppelA AdditioneDeuxNombre ReadNextRecord Appel_2

Voici quelques mauvais exemples

Août // Lettre accentuée 1_Appel // Débute par un chiffre

Appel-Sp // Le tiret est interdit

Attention : on ne doit pas utiliser comme identificateurs les mots réservés ou les noms de procédures et de fonctions définis par le langage.

Les déclarations

Les déclarations sont constituées de deux éléments principaux :
les constantes (CONST) et les variables (VAR) .
Ces deux éléments peuvent être écrits dans n'importe quel ordre et répétés plusieurs fois.

Les déclarations sont regroupées sous l'un des deux éléments ci-avant, qui joue le rôle de titre. Chaque déclaration se termine par un point-virgule.

Exemple :

```
CONST
   MaxNum = 1235;
   Chaine   = 'Bonjour';
Var
        I,J   : Integer;     // *** Variables non initialisée *
        K,L : Word=1;    // *** Variables initialisée *
```

Les types booléens

En automatisme la dénomination Bit correspond à Boolean sous Visual I/O.
Cependant un bit d'un mot fait partie aussi des booléens de Visual I/O.

Les types booléens ne peuvent prendre que deux valeurs prédéfinis, FALSE et TRUE.

Les valeurs ordinales de FALSE et TRUE, sont respectivement 0 et 1.
Sortie : BOOLEAN

Bit dans un mot

Si I0 est du type Integer I0.4 est du type Boolean dans Visual PLC. Exemple `I0.4 :=True`

Les types tableaux

Un type tableau (ARRAY) est une structure qui regroupe des éléments de même type.
Voici la syntaxe des types tableaux :

```
PAIE        : Array [1..4] Of REAL
SEMESTRE    : Array [1..6] Of BOOLEAN
```

Pour accéder à un élément du tableau, on passe par une indexation, par exemple Paie[2], dans ce cas on intervient sur le deuxième nombre réel du tableau PAIE.

Les types chaînes de caractères

Un type chaîne de caractères (STRING) est une suite de caractères dont le nombre peut varier de 1 à 255. On peut assimiler ce type à un tableau de 256 éléments. Le premier élément, indice 0, contient la longueur utile de la chaîne.

Voici la syntaxe de types chaîne :

```
Chaine      : STRING
Ich         : STRING ='Automatique';  // ** Chaîne initialisée *
```

L'index 0 renvoie la longueur de la chaîne, qui est une valeur ASCII.

Une chaîne de caractères s'écrit entre deux apostrophes. Si une apostrophe entre dans la composition de la chaîne de caractères, elle est doublée.

Exemple :

```
Var
  Msg : STRING;

BEGIN
   Msg : = 'Bonjour';
   Msg := 'Aujourd''hui' :
END;
```

Pour connaître la longueur d'une chaîne, on dispose de la fonction prédéfini de Pascal Length(Msg). On peut aussi utiliser la valeur ordinale de l'indice 0 de la chaîne Ord(Msg[0]), qui renverra 7 dans l'exemple 'Bonjour'.

Si l'on concatène une chaîne de 200 caractères et une chaîne de 100 caractères, les 45 derniers caractères seront perdus.

Les opérateurs

Voici les opérateurs :

Not, /, Mod, And, Shl, Shr +, -, Or, Xor, =, <>, >, <, <=, >=

Certains de ces opérateurs vont avoir la lourde tâche de réaliser plusieurs sortes d'opérations.

Les opérateurs arithmétiques

Les opérateurs arithmétiques réalisent les quatre opérations élémentaires sur les nombres. Ce sont l'addition, la soustraction, la multiplication et la division représentées par les symboles habituels de ces opérations, hormis la multiplication représentée par l'astérisque *. Un autre opérateur est utilisé pour la division l'opérateur MOD, celui-ci reverra le reste de la division.
Visual PLC simplifie au maximum le calcul arithmétique. Vous pourrez mélanger les types. Ainsi la division d'un mot de 32 bits (Integer) par un Real est permise. Le compilateur transtype automatiquement l'expression arithmétique lorsqu'il trouve des types différents dans une équation.
Attention aux arrondis dans ce cas !

La **division par 0 ne plante pas** le programme elle renvoie le résultat 0.

L'opérateur chaîne de caractères

Le seul symbole opérateur utilisé par les chaînes de caractères est le +, qui sert à concaténer des chaînes, des caractères ou des chaînes compactées. Dans tous les cas, la concaténation de chaînes est limitée à la longueur de 255 caractères.

Voici un exemple :

```
TS :='Visual PLC'+' Temps Réel';
```

Le résultat dans TS est : 'Visual PLC Temps Réel'

Pour tester ou affecter un élément d'une chaîne de caractères, utilisez la syntaxe suivante.
Chaîne[X]

Exemple:

```
If TS[1]='v' Then Ts[1]:='V';
```

Les opérateurs logiques

Les opérateurs logiques donnent des résultats entiers avec des opérandes entiers.

- And et binaire
- Or ou binaire
- Xor ou exclusif binaire
- Shl décalage de bits à gauche
- Shr décalage de bits à droite

Voici la table de vérité de AND OR ET XOR :

SHL et SHR réalisent des décalages de bits à droite ou à gauche sur les nombres entiers.
Exemple, I:= 5 SHL 1 donne comme résultat 10.
L'octet de départ en binaire est 0000 0101, après le décalage l'octet devient 0000 1010. Le bit le plus à gauche est perdu.
Par contre, si le décalage avait été SHR 1, le résultat aurait donné : 0000 0010.
Le bit le plus à droite est perdu.

Les opérateurs booléens

Avec les opérateurs booléens sont utilisés des opérandes booléens.

Tableau des opérateurs booléens

- And & et logique (le caractère d'imprimerie & est valide comme opérateur logique)
- Or ou logique
- Xor ou exclusif
- Not négatif logique

Exemple :
Etat:= I0.0 and Cde & Not(W0.12);

Trucs et astuces

Affectation d'un résultat à un Boolean

```
Var
  V     : Integer ;
  Egal  : Boolean ;
Begin
   V :=12 ;
   Egal :=(V=12);     // *** Résultat un booléen qui est affecté à Egal *
End ;
```

Les opérateurs sur les ensembles

Les opérateurs relationnels sur les ensembles sont au nombre de quatre.

Tableau des opérateurs relationnels d'ensembles

- = égal.
- <> différent.
- <= Inférieur ou égal.
- >= Supérieur ou égal.

Les instructions

Les instructions sont des actions exécutées à l'intérieur d'un bloc dans l'ordre de programmation.

ON distingue les instructions simples :

Affectations, appel de procédures ou de fonctions.

et les instructions composées :

IF, CASE, REPEAT, WHILE, FOR.

Les instructions simples

Les instructions simples sont au nombre de deux :

- Les instructions d'affectation.
- Les instructions de procédure ou de fonction.

Les affectations

L'instruction d'affectation := permet de charger une variable par une nouvelle valeur. Celle-ci est une expression d'un type compatible avec la variable.

Voici quelques exemples :
TS :='Bonjour' ;
I :=12 ;
R :=12.3 ;

Appel d'une procédure

Une telle instruction exécute une procédure. Celle-ci est nommée par son identificateur et, éventuellement, par une liste d'arguments (paramètres formels).

Voici la syntaxe :
If Button **Then Initialise**;

Le nombre et le type des paramètres passés en arguments de procédure doivent correspondre à ceux décrits dans l'en-tête de la procédure appelée.

Les instructions composées

Les instructions composées sont constituées d'instructions conditionnelles (IF, CASE) d'instructions répétitives (REPEAT, FOR, WHILE).

Instruction IF THEN ELSE

Dans Visual PLC l'instruction IF THEN ELSE est souvent utilisée.
La Syntaxe est IF *QuelqueChose* THEN *Action* ELSE *Autre action ;*

1er Cas
```
If Compteur=10 Then Compteur:=0; // Ici le point virgule doit être mis derrière 0
```

Si la variable compteur est égal à 10 alors Compteur prend la valeur 0.
Dans ce cas Else n'est pas utilisé, vous devez mettre ; derrière la dernière instruction.

2ème Cas
```
If Compteur=10 Then Compteur:=0    // pas de ; derrière 0 cas Else est présent
               Else Compteur:=Compteur+1; // dernière instruction alors ;
```

Si la variable compteur est égal à 10 alors Compteur prend la valeur 0 Sinon incrémentation de compteur.
Dans ce cas pas de ; à la fin de la première ligne car un Else est utilisé dans la deuxième ligne.
3ème cas :
```
If Compteur=10 Then
Begin
  Compteur:=0;
  Ok:=True;
End;  // *** dernière instruction alors ; ***
```
Si compteur égal 10 alors exécution d'une série d'instructions contenues dans un bloc Begin End;
Dans ce cas un ; est présent derrière End car aucun Else n'est utilisé.

4ème cas :
```
If Compteur=10 Then
Begin             // Premier Bloc d'instructions
   Compteur:=0;
   Ok:=True;
End Else          // pas de ; derrière End car un Else est présent
Begin             // Deuxième bloc d'instructions
    Compteur:=Compteur+1;
    Ok:=False;
End;              // dernière instruction alors ; ***
```
Si compteur égal 10 alors exécution d'une série d'instructions contenues dans un bloc Begin End; Sinon exécution d'un autre bloc d'instructions.

Instruction REPEAT UNTIL

La Syntaxe est REPEAT *QuelqueChose* UNTIL *condition Vraie;*
Le mot Repeat permet de marquer le début du bloc d'instructions et Until la fin de celui-ci.

Exemple :
```
Repeat
   Compteur:=Compteur+1;
   Ok:=False;
Until Compteur=10;      // *** ; obligatoire
```

La syntaxe suivante est aussi correcte mais inutile. Le Begin et le end sont inutile.
```
Repeat
   Begin
      Compteur:=Compteur+1;
      Ok:=False;
    End;
Until Compteur=10;      // *** ; obligatoire
```

L'instruction Repeat Until est voisine de l'instruction While Do.

Cependant dans Repeat Until le bloc d'instructions est exécuté au moins une fois.
Car le test est en fin de bloc.

Instruction WHILE DO

La syntaxe est WHILE *Condition Vraie* DO *QuelqueChose;*
Tant que la condition est vraie alors exécution d'une instruction ou d'un bloc d'instructions.

Exemple :

```
While Compteur<10 Do
Begin
   Compteur:=Compteur+1;
   Ok:=False;
End;        //*** ; obligatoire ******
```

While Compteur<10 **Do** Compteur:=Compteur+1;
L'instruction While Do est voisine de l'instruction Repeat Until.
Cependant dans While Do le bloc d'instructions n'est forcément exécuté.
Car le test est en début de bloc.

Instruction CASE OF

Le complément de l'instruction IF, qui n'offre que deux possibilités de choix, est l'instruction CASE, qui permet un choix multiple.
Celle-ci consiste en une expression de type scalaire dont les bornes sont celles du type Non flottant.

Case of permet de tester une variable entière et d'offrir des choix en fonction de celle-ci.
A un mot Case doit correspondre un End; L'instruction Else est facultative.

```
Exemple :
Begin
   Case Indice of
            1     : Consigne:=11;
            2     : Consigne:=22;
            3     : Consigne:=11;
            100   : Consigne:=100;
            Else  Consigne:=0;    // *** Else est facultatif
      End;
End;
```

Si indice=1 alors Consigne reçoit la valeur 1teste Indice=2 et ainsi de suite. Si aucun test n'est vrai alors l'instruction facultative Else est exécutée et consigne est affectée de la valeur 0. Si un test est vrai les autres tests ne sont pas exécutés.

Instruction FOR TO /DOWNTO

L'instruction FOR permet d'exécuter un ensemble d'instruction d'une façon répétée en modifiant une variable qui sert de compteur de boucle.
L'avantage des boucles FOR est qu'on ne risque pas de se retrouver dans un processus infini qui oblige à relancer l'ordinateur.
Des boucles FOR peuvent s'imbriquer. Exemple d'une table de multiplication :
Les boucles FOR peuvent incrémenter une variable For To
Les boucles FOR peuvent Décrémenter une variable For DownTo
La syntaxe est :

For .. **To/Downto** … **Do**

Exemple :

```
      For I :=1 to 10 do Somme:=Somme+Tablo[I];
```

Cette boucle For évolue de 1 à 10.

Exemple :

```
      For I :=10 DownTo 1 do Somme:=Somme+Tablo[I];
```

Cette boucle For évolue de 10 à 1.

Procédures et fonctions

Les procédures et fonctions peuvent être considérées comme des sous-programmes qui effectuent des tâches élémentaires. Elles sont nécessaires à la bonne compréhension d'un programme et à leur maintenance. Les procédures et fonctions peuvent être déclarées dans la partie Librairies VPU et peuvent-elles mêmes utilisées d'autres fonctions dans d'autres VPU.

Structure

Les procédures et les fonctions sont constituées d'un en-tête et d'un bloc.

L'en-tête est constitué des mots réservés PROCEDURE ou FUNCTION, suivis d'un identificateur, éventuellement d'une liste d'arguments entre parenthèses, et pour les fonctions, du type du résultat renvoyé.

Voici la syntaxe :
Procedure MonProc1 (Element : String) ;
Procedure MonProc2 (**Var** Element : String) ;

Function MaFunc (Nombre : Real) : Boolean ;
Dans le cas de la procédure MonProc1 la variable Elément est passée à la procédure.
Cette Variable ne peut être seulement lue.
Dans le cas de la procédure MonProc2 la variable Elément est passée à la procédure.
Cette Variable ne peut être lue mais aussi écrite.
Dans le cas de la fonction MaFunc la variable Nombre est passée à la fonction.
Cette variable ne peut être que lue et non écrite.
En retour cette fonction renvoi un booléen qui peut être testé ou non par la l'instruction l'appelant.

Renvoi d'un résultat par une Fonction :

Une fonction renvoi un résultat qui est typé (Boolean, Real, Etc..).
Ce résultat est une variable créée par la fonction elle-même. Cette variable résultat est mot réservé du langage il s'appelle **RESULT**.

Exemple :

```
Function Addition ( V1,V2 : Real) : Real ;
Begin
   Result :=V1+V2 ;
End ;
```

Result : est la variable de retour de la fonction. Dans l'exemple ci-dessous total recevra le résultat de la fonction addition en retour.
Total :=Addition(10,20) ;

Paramètres optionnels dans les procédures et les fonctions

A partir de la version 7 de Visual I/O, les procédures et fonctions peuvent avoir des paramètres optionnels.

Exemple :

Considérons une nouvelle procédure de coloriage de qui permettrai de colorier la figure courante ou une figure précise :

Syntaxes d'appels possibles :

```
      SetfillColorEx( CLRED);    ou
      SetfillColorEx( CLRED,'RECTANG1');

Procedure SetFillColorEx( Color : Integer; Fig : String);
Begin
  If Fig<>'' Then Commut_Fig(Fig); // si le paramètre est renseigné on passe sur
                                   // la figure
  SetFillColor(Color); // appel de la procédure de coloriage de la
                       // figure en cours
End;
```

Ici si le paramètre FIG n'est pas renseigné , FIG prends alors la valeur chaîne vide.
Cas d'un paramètre optionnel Integer celui-ci prends la valeur 0.
Cas d'un paramètre optionnel Booléen celui-ci prends la valeur False.

☞ **Note** : Des exemples sont consultables dans la librairie Bonus32.pas

Jeu d'instructions de base du langage pascal pour Visual I/O

Le jeu d'instruction incorpore évidement les instructions composées décrites précédemment :
IF/THEN/ELSE REPEAT/UNTIL FOR/TO/DOWNTO CASE

Les instructions de base sont les suivantes :
Opérations Arithmétiques + - * /
Opération Logiques OR AND XOR

Shl Décalage à gauche de X bits
```
I:=J SHL 3;
```
Shr Décalage à droite de X bits
```
I:=J SHR 3;
```
Not Inverse un boolean
```
I0.0:=Not(I0.1);
```
IntToStr Converti un Integer en Chaîne de caractères
```
TS:=IntToStr(12);
```
StrToInt Converti une en Chaîne de caractères en un integer
```
I:=StrToInt('12');
```
FloatToStr Converti un Real en chaîne de caractères
```
TS:=FloatToStr(12.45);
```
IntToHex Converti un entier en Chaîne représentant un nombre Hexadécimal
```
TS:=IntToHex(I,Digits);
```
Move Transfert une série d'octets consécutifs dans une autre série
MOVE(Source,Destination,Nombre);
Copy Copy une partie d'une chaîne dans une autre chaîne de caractères.
```
TS:=Copy('Arsoft',3,4);   // *** TS:='soft' **
```
Val Converti une chaîne de caractères en numérique.
```
Val(TS,I,Ok);   // I=Real  OK=Integer); Si OK=0  I=valeur correcte
```
Pos Renvoi la position d'un texte dans une chaîne de caractères.
```
I:=Pos('soft',TS);    // ** I=3 si Ts='Arsoft' *
```
Delete Efface une portion de la chaîne spécifiée
```
Delete(TS,3,4);  // ** TS=Arsoft → TS=Ar *
```
Insert Insert une chaîne dans une autre chaîne
```
Insert(TS,'Visual Plc ',1); // TS='Arsoft' -→ TS='Visual PLC Arsoft'
```
Length Renvoi la longueur de la chaîne de caractères
```
I:=Length('Arsoft');  // ** I=6 *
```
Addr,@ Renvoi l'adresse mémoire d'une variable
```
A:=Addr(Compteur); ou A:=@(Compteur);
```
Chr Renvoi le caractère associé au code AscII .
```
TS:=Chr(65);   // *** TS='A' **
```
Ord Renvoi le code Ascii du caractère spécifié.
```
I:=Ord('A');  // *** I=65 **
```
FillChar Rempli une zone de la valeur spécifiée.
```
FillChar(Tablo,16,0); // Tablo : array [1..4] of integer Les 4
                      // premiers éléments du tableau sont mis à 0
```
Cos Renvoi le cosinus de l'argument passé en radians
```
R := Cos(Pi);
```
Sin Renvoi le sinus de l'argument passé en radians
```
R := Sin(Pi);
```
Exp Renvoie la valeur exponentielle de X.
```
E := Exp(1.0);
```
Int Renvoie la partie entière d'un nombre réel.
```
R := Int(123.456);  //123.0
```
Frac Renvoie la partie décimale d'un réel.
```
R := Frac(123.456); //0.456
```
Trunc Tronque un réel en entier.
DegToRad Converti des degrés en radians
```
R:=Cos(DegToRad(60));
```
Abs Renvoie une valeur absolue.
```
r := Abs(-2.3);   //  2.3
```
ArcTan Calcule l'arctangente de la valeur donnée.
```
R := ArcTan(Pi);
```
Ln Renvoie le logarithme naturel d'une expression réelle.
```
E := Exp(1.0);
```
Sqr Renvoie le carré d'un nombre.
Sqrt Renvoie la racine carré d'un nombre.
LoWord Renvoie la partie faible 16 bits d'un integer.
```
W:=Loword(I);
```
HiWord Renvoie la partie fort 16 bits d'un integer.
```
W:=Hiword(I);
```
Lo Renvoie les poids faibles d'un integer
```
W:=Lo(I);   // 0..255}
```
Hi Renvoie les poids forts d'un integer.
```
W:=Hi(I);   // 0..255}
```
Port En lecture renvoie la valeur d'un port Physique D'E/S
```
V:=Port($2F8);
```
En écriture affecte d'une valeur un port Physique D'E/S
```
Port($2F8):=0;
```
Mem En lecture renvoie la valeur d'une adresse mémoire
```
V:=Mem($C800,0);       // ** lecture d'un byte *
```
En écriture affecte d'une valeur une adresse mémoire
```
Mem($C800,0):=255;    // ** écriture d'un byte
```

Procedure **ClearBlank** Efface les espaces avant et après la chaîne de caractères spécifiée.
Exemple : *TS :=Clearblank(TS1) ; // TS reçoit TS1 sans les Blancs*
Function **StrChange** (Ch : String) : Boolean; Renvoi Vrai si la chaîne de caractères à changée de contenu.
Exemple : *If StrChange(MaChaine) Then Addstring(MaChaine);*
Procedure **HardCopy**; *Permet de faire une copie d'écran sur imprimante.*
Exemple : *If Button then HardCopy;*
Procedure **MoveFig** ; // *Permet de déplacer sur l'écran à l'aide de la souris n'importe quelle figure.*

// Permet par programme d'animer un bouton en bouton poussoir simple.
Procedure **PushButton** (State : Boolean ; Key : Word) ;
*// Permet de mettre à l'échelle en fonction de bornes d'entrées et de sorties **
Function **Scale** (V : real ; InputLowRange,InputHighRange, OutputLowRange,OutputHighRange : Real) : Real ;

Exemple : R :=Scale (Eana1,0,2048,-10,100) ;
Eana varie de 0 à 2048 et la sortie équivalente de –10 à 100. R recoit la conversion.
☞ **Note** : Cette function fonctionne aussi avec des integers (transtypage par Visual I/O automatique).

// Permet de donner la main plus rapidement à d'autres applications Windows lancées en parallèle de Visual I/O.
Procedure **PROCESSMESSAGES** *// Dispatch les messages Windows*.
Procedure **Beep** *// Emet un Bip sur le haut parleur du PC.*
Procedure **PlaySound** (WavFile : String) ; *// Joue le fichier d'extension .WAV spécifié.*

Instructions concernant les tracés simples (cercles, ellipses, rectangles ...)

// Renvoi sur 1 tour de cycle l'état d'un bouton poussoir simple ou d'une figure servant de bouton}
Function **Button**:Boolean;
Procedure **ExclusiveButton** ; *// Sélection exclusive sur des boutons groupés}*
Procedure **Filled**; *// Remplie une figure avec la couleur de remplissage}*
Procedure **UnFilled**; *// Rend une figure transparente}*
Procedure **FirstColors**; *// Restitue la couleur de création d'une figure}*
Function **GetButton**: Boolean; *// Retourne l'état d'un bouton}*
Function **GetAngle**:Integer; *// Retourne la valeur angulaire d'un Polygone}*
Function **GetColor**:LongInt; *// Retourne la couleur de contour d'une figure}*
Function **GetFillColor**:LongInt; *// Retourne la couleur de remplissage d'une figure}*
Function **GetPenWidth**:Integer;*// Retourne l'épaisseur de tracé de contour d'une figure}*
Function **GetXFin**:Integer; *// Retourne l'abscisse droite d'une figure}*
Function **GetXorg**:Integer; *// Retourne l'abscisse gauche d'une figure}*
Function **GetYFin**:Integer; *// Retourne l'ordonnée droite d'une figure}*
Function **GetYorg**:Integer; *// Retourne l'ordonnée gauche d'une figure}*
Function **IsFilled**:Boolean; *// Indique si une figure est remplie}*
Procedure **SetAngle**(Angle:Integer); *// Fixe la valeur la valeur Angulaire d'un polygone*
Procedure **SetButton**(On_Off:Boolean); *// Force l'état d'un bouton}*
Procedure **SetColor**(Color:LongInt); *// Fixe la couleur de contour d'un figure}*
Procedure **SetFillColor**(Color:LongInt); *// Fixe la couleur de remplissage d'une figure}*
Procedure **SetPenWidth**(Width:Integer); *// Fixe l'épaisseur de tracé de contour d'un figure}*
Procedure **SetXFin**(XEnd:Integer); *// Fixe l'abscisse droite d'une figure}*
Procedure **SetXorg**(XOrg:Integer); *// Fixe l'abscisse gauche d'une figure}*
Procedure **SetYFin**(YEnd:Integer); *// Fixe l'ordonnée droite d'une figure}*
Procedure **SetYorg**(YOrg:Integer); *// Fixe l'ordonnée gauche d'une figure}*
Procedure **FloodFill** (Flood : boolean; Color : Integer ; Percent : Integer) ;
// Rempli un polygone
Flood = True la figure se remplie sinon vide
Color = couleur de remplissage
Percent = Pourcentage de remplissage
// Permet de créer un bouton à accrochage sur toute figure
Procedure **ToggleButton** (Var State : Boolean ; Key : Integer) ;
Procedure **ValidButton** (State : Boolean) ; *// Permet de valider ou non un bouton}*
// Transforme une figure en potentiomètre vertical
Procedure **SliderVertical** (Var Variable : Real ;PixelMax : Integer ; LowRange, HighRange : Real) ;
// Transforme une figure en potentiomètre horizontal
Procedure **SliderHorizontal** (Var Variable : Real ;PixelMax : Integer ; LowRange, HighRange : Real) ;

Instructions concernant les ListBox et les ComboBox

```
Function    AddString (S:String): Integer;        // Ajoute une ligne dans une Listebox ou un ComboBox
Procedure ClearList:                              // Efface le contenu d'une ListeBox ou ComboBox
Procedure ClearModify:                            // Met à zéro le flag interne de modification d'une zone de saisie
Procedure ComboHideList:                          // Cache la liste déroulante d'un ComboBox
                                                  // Sélectionne le texte dans la zone de saisie d'un ComboBox
Function ComboSetEditSel(Var StartP,EndP:Integer):Boolean;
Procedure ComboShowList;                          // Affiche la liste déroulante d'un ComboBox
// Supprime une ligne dans une  ListeBox ou ComboBox
Function DeleteLine(LineNumber:Integer):Boolean;
Function DeleteSelection:Boolean;  // Efface les caractères sélectionnés  dans une zone de saisie Multiligne
Function DeleteString (LineNumber:Integer):Integer;  // Efface une ligne dans un  ComboBox ou une listeBox
Function GetCount:Integer;       // Renvoi le nombre d'éléments d'une listeBox ou d'un  ComboBox
Function GetHwnd:Integer ;       // Renvoi le Handle de la fenêtre contenant cette  instruction
Function GetSelCount:Integer; // Renvoi le nombre de sélection dans une listeBox multiselect ou non
// Renseigne sur l'état d'un élément dans une listeBox multiselect ou non
 Function GetSelected (Item:Integer):Boolean;
// Renvoi le numéro de ligne sélectionné dans une  listeBox ou un  ComboBox
Function GetSelindex:Integer;
// Renvoi la chaîne sélectionnée dans une listeBox ou un  ComboBox
Function GetSelString:String;   .
// Renvoi le contenu de l'élément pointé dans une ListeBox ou un  ComboBox            .
Function GetString (LineNumber:Integer):String;
// Renvoi le nombre de caractères contenus dans une  zone de saisie d'une EditBox ou d'un ComboBox
Function GetTextLen:Integer;
Function GetValue:Real;                           // Renvoi la valeur d'une valeur affichée à l'écran.
// Permet d'insérer une chaîne de caractère à un emplacement donné  dans une listeBox ou un  ComboBox.
Function InsertString(S:String,Position:Integer):Integer;
Function IsModified:Boolean;   // Indique si le champ d'une zone de saisie a été modifié
Procedure LDir(Mask:String);   // Remplit une liste de noms de fichiers dans une listeBox ou un ComboBox
Procedure LDirEx(Mask:String);  //  Affiche l'extension dans la ListBox
 // Fixe le numéro de ligne sélectionné dans une listeBox ou un ComboBox
Function SetSelIndex(LineSelected:Integer):Integer;
// Sélectionne une ligne dans une listeBox ou un ComboBox
Function    SetSelString (S:String,Pos:Integer):Integer;
Procedure SetText(S:String);  // Force un texte dans une zone de saisie d'une EditBox ou d'un  ComboBox
Procedure SetValue(Value:Real);  // Fixe la valeur d'une figure valeur affichée à l'écran
Function StrListExist(String:S):  // Indique si la chaîne spécifiée existe dans la listeBox ou le ComboBox.
// Retourne le nom d'un bitmap affiché dans une ListBox ou un  ComboBox
Function VisuBmp(NumLine:S):String;
// Teste si une chaîne de caractères existe dans la ListBox ou le ComboBox
Function StrListExist ( Chaine : String) : Boolean ;
```

Instructions concernant les Cases à Cocher et Radio Boutons

```
Procedure Check;                 // Coche une case à Cocher
Procedure UnCheck;               // Décoche  une case à Cocher
Procedure Exclusive;             // Permet d'avoir une sélection exclusive sur des cases à cocher  groupées.
Function   Get_Focus:Boolean;                  // Indique si le contrôle Windows a le focus
Function   GetCheck:Integer;                   // Renvoi l'état d'une case à cocher 0/1.
Procedure SetCheck(Flag:Integer);              // Fixe l'état d'une case à cocher.
Procedure SetRadio( State : Boolean) ;         //  fixe l'état d'un radio bouton
Function   GetRadio : Boolean ;                //  Renvoi l'état d'un Radio-Bouton
Function   IsModified:Boolean;                 //  Indique si le contrôle Windows est modifé
Function  GetCheckBox : Boolean;               // Renvoi l'état de la case à cocher.
Procedure SetCheckBox (state : Boolean);       // Fixe l'état de la case à cocher.
```

Instructions concernant les zones d'édition

Procedure **Clear**; *// Efface la Zone de Saisie d'un Combo ou d'une zone d'édition.*
Procedure **ClearModify**; *// Met à zéro le flag interne de modification d'une zone de saisie}*

// Supprime une ligne dans une zone de saisie multilignes, une ListeBox ou ComboBox
Function **DeleteLine** (LineNumber:Integer):Boolean;

Function **DeleteSelection**:Boolean; *// Efface les caractères sélectionnés dans une zone de saisie Multiligne*
// Renvoi vrai quand l'opérateur frappe <Entrée> dans une zone de saisie d'une EditBox ou d'un ComboBox (Type 1 ou 2)
Function **EditCR**:Boolean;

// Renvoi vrai quand l'opérateur frappe <Echap> dans une zone de saisie d'une EditBox ou d'un ComboBox (Type 1 ou 2).
Function **EditESC**:Boolean;

// Renvoi la ligne demandée dans une zone de saisie multilignes
Function **GetLine**(Var S:String,Line:Integer):Boolean;

// Renvoi la ligne contenant le caractère à une position donné.
Function **GetLineFromPos**(CharPos:Integer):Integer;

// Renvoi la longueur d'une ligne dans une zone de saisie multilignes.
Function **GetLineLength**(LineNumber:Integer):Integer;

// Renvoi le nombre de lignes existantes dans une zone de Saisie Multilignes.
Function **GetNumLines**:Integer;

// Permet d'ajouter une chaîne de caractères dans une ListeBox,ComboBox ou Editbox multilignes
Function **AddString**(S:String):Integer;

// Renvoi la position du curseur ou de la sélection dans une EditBox.
Procedure **GetSelection**(Var StartPosition,EndPosition:Integer);

// Renvoi le contenu d'une zone de saisie d'une EditBox ou d'un ComboBox.
Function **GetText**:String;

// Renvoi le nombre de caractères contenus dans une zone de saisie d'une EditBox ou d'un ComboBox.
Function **GetTextLen**:Integer;

Function **GetValue**:Real; *// Renvoi la valeur d'une valeur affichée à l'écran.*
Procedure **Editinsert**(S:String); *// Permet d'insérer dans une zone de saisie une chaîne de caractères.*

// Permet d'insérer une chaîne de caractère à un emplacement donné dans une listeBox ou un ComboBox
Function **InsertString**(S:String,Position:Integer) : Integer;

// Indique si le champ d'une zone de saisie,case à cocher radio combo a été modifié
Function **IsModified** : Boolean;

Procedure **UnCheck** ; *// Décoche une case à cocher .*

// Fixe l'attribut d'une zone de saisie (Editbox), soit en lecture seule ou en lecture et écriture
Procedure **SetReadOnly** (Readonly:Boolean);
// Fixe la position du curseur ou de la sélection dans une EditBox.
Function **SetSelection**(StartPosition,EndPosition:Integer):Boolean;

// Force un texte dans une zone de saisie d'une EditBox ou d'un ComboBox.
Procedure **SetText**(Text:String);

Procedure **SetValue**(Value:Real); *// Fixe la valeur d'une figure valeur affichée à l'écran.*
Function **LooseFocus** : Boolean ; *// Teste si le champ de saisie permet le focus*
Procedure **NextFocus** ; *// Passe au champ de saisie suivant*

Instructions Systèmes (Gestion clavier, souris, programmes...)

Procedure **ActiveDVE**; *// Permet activer les synoptiques de type Dialog pour un tour de cycle*
Function **WinApparition**:Boolean ; *// Indique si le synoptique vient d'apparaître}*
Function **Disparition** : Boolean ; *// Indique si le synoptique vient de disparaître*
// Fixe le contour d'une fenêtre interne incluse dans le synoptique}
Procedure **SetBorderStyle** (Border : Word) ;
Function **GetParentName** : String ; *// Renvoie le nom du parent de la figure*
// Fixe les tailles mini et maxi du synoptique en cours (Obsolète dans éditeur caractéristiques syno)
Procedure RestrictWinSize (WidthMin, HeightMin, WidthMax, HeightMax : Integer) ;

Procedure **WinBKGNDColor** (Color : Integer) ; *// Fixe la couleur de fond du Synoptique*
// Autorise ou non le rafraîchissement graphique du synoptique
Procedure **EnableRefresh** (Refresh : Boolean) *;*
Procedure **AllIconic**; *// Met toutes les fenêtres présentes à l'écran sous forme d'icônes.*
Procedure **SetCanClose** (**CanClose** : Boolean); // Autorise la fermeture ou non du synoptique

Variable **AltActived : Boolean;** *// Variable contenant l'état de la touche Alt.*
Function **Arrive**:Boolean; *// Test si la Souris arrive dans la figure concernée.*
Function **BlinkN**:Boolean; *// Renvoi true tous les 500 ms.*
Function **BT100**:Byte; *// Renvoi 1 ou 0 les 100 ms.*
Function **BT_500**:Byte; *// Renvoi 1 ou 0 les 500 ms.*
Procedure **CloseAppli**; *// Ferme l'application ou le synoptique courant.*
Function **Commut_Fig** (DrawingName : String):Boolean; *// Permet le Changement de figure.*
Function **GO2**(DrawingName:String):Boolean; *// Permet le Changement de figure équivalent à commut_Fig*

Function **ControlVisible**: *// Indique si la figure est visible ou non.*
Variable **CtrlActived** *// Variable contenant l'état de la touche Ctrl.*
Procedure **CursorNormal**; *// Affiche le curseur de la souris sous sa forme normale*
Procedure **CursorWait**; *// Affiche le curseur de la souris sous forme d'un sablier.*

Variable **DBClick** *// Indique si un DoubleClick souris à été effectué.*
Function **Depart**:Boolean; *// Test si la Souris sort de la figure concernée.*
Procedure **Displace**(DX,DY:Integer); *// Déplace graphiquement une figure en relatif.*
Procedure **ODisplace**(DX,DY:Integer); *// Déplace graphiquement une figure en absolu*
Procedure **OEnLarge**(DX,DY:Integer); *// Agrandi graphiquement une figure en absolu.*

Procedure **Display**(Name:String); *// Affiche un Syno ou lance un exécutable.*
Function **FormVisible** (FormName : String) : Boolean; // renvoie True si le synoptique de nom contenu dans le paramètre Formname est affiché.

// Affiche un Syno ou lance un exécutable et attend sa fermeture
Procedure **DisplayModal**(Name:String);
Function **DoubleClick**:Boolean; *// Permet de tester un Double-Click souris dans une figure.*

// Permet de stocker la chaîne de caractères servant de transfert lors d'une opération de drag and Drop
Procedure **Drag**(S:String);

Function **Drop** : Boolean; *// Renseigne sur la possibilité de drop (en Drag & Drop) dans la figure*
Variable **End_Visu : Boolean;** *// Le dernier tour de cycle du programme avant fermeture de l'appli*

Procedure **Execute** (ExeName:String); *// Lance un exécutable.*
Procedure **ModalExecute** (ExeName:String); *// Lance un exécutable et attend sa fermeture.*

Function **GetCaption** : String; *// Renvoi le texte contenu dans la barre de titre de la fenêtre.*
Function **DragFrom** : String; *// Renvoi le Nom de la figure qui a déclenché le drag & drop.*
Function **GetDrop** :String; *// Renvoi la chaîne interne de drag and Drop*

Function **GetHwnd** : Integer; *// Renvoi le Handle de la fenêtre contenant cette instruction*
Function **GetKey** : Char; *// Renvoi la touche frappée au clavier*
Function **GetMainHwnd**:Integer; *// Renvoi le Handle de la fenêtre synoptique*

Function **GetNDecimales**:Byte; *// Renvoi le nombre de décimales affichées d'une valeur numérique.*
Function **GetPage**:Integer; *// Renvoi la page en cours de visualisation dans un synoptique (obsolete).*

// Affiche une palette de couleur pour le choix d'une couleur
Function **GetPaletteColor**(Var Color:Integer):Boolean;
Function **GetTextMessage**:String; *// Renvoi le message affiché sur un bouton ou dans une figure texte*

Function **GetFigName** : String ; *// Renvoi le nom de la figure*
Function **GetVirKey** : Word; *// Renvoi le code des touches virtuelles frappées.*
Procedure **HideControl**; *// Permet de cacher une figure*
Function **Is_Me** : Boolean; *// Indique si la souris est dans la figure*
Function **KeyUp** : Boolean; *// Test si une Touche du clavier a été relâchée*
Function **LButtonDown**:Boolean; *// Test si le bouton gauche souris à été enfoncé sur le synoptique*
Function **LButtonUp**:Boolean; *// Test si le bouton gauche de la souris à été relâché sur le synoptique*
Function **LoCaseS** (S:String):String; *// Renvoi la chaîne passée en argument en minuscules*
Function **UpCaseS**(S:String):String; *// Renvoi la chaîne passée en argument en majuscules.*
Procedure **SetNDecimales** (Decimales : Integer); *// Fixe le nombre de décimales d'une valeur numérique*
Function **GetNDecimales** : Integer; *// renvoie le nombre de décimales d'une valeur numérique*

Procedure **NormalizeTopMosts**; *// Passe en arrière toutes les fenêtres affichées à l'écran*
Procedure **RestoreTopMosts** ; *// Remet les fenêtres Windows en avant plan suite à NormalizeTopMosts*

Function **Previous_Group** : Boolean ; *// Passe sur la figure suivante groupée*
// Test si le bouton droit de la souris à été enfoncé quelque part sur le synoptique
Function **RbuttonDown** : Boolean ;
// Test si le bouton droit de la souris à été Relâché quelque part sur le synoptique
Function **RbuttonUp** : Boolean ;
Procedure **SetCaption** (Title : String); *// Fixe un Texte dans la barre de titre du synoptique*
Procedure **SetPage**(Page : Integer); *// Force le passage à une page sur le synoptique*
Variable **ShiftActived** : Boolean ; *// Renvoi l 'état de la touche Shift du clavier*

Procedure **ShowControl** ; *// Fait apparaître un élément graphique caché du synoptique*
Variable **Start_Visu** : Boolean ; *// Indique le tout premier affichage du synoptique*

Function **TopLBDown**:Boolean; *// Test si le bouton gauche de la souris à été enfoncé sur la figure*
Function **TopLBUp**:Boolean; *// Test si le bouton gauche de la souris à été relâché sur la figure*
Function **TopRBDown**:Boolean; *// Test si le bouton droit de la souris à été enfoncé sur la figure.*
Function **TopRBUP**:Boolean; *// Test si le bouton droit de la souris à été relâché sur la figure.*

Function **TypeFig** : Integer; *// Renvoi le type de figure. Voir codes dans Windows.pas*
Variable **ValidXScroll** (State : Boolean); *// Affiche la barre de scroll en X sur le synoptique*
Variable **ValidYScroll** (State : Boolean); *// Affiche la barre de scroll en Y sur le synoptique*

// Affiche une petite fenêtre de fond jaune contenant un message sous la figure
Function **WinMess** (Message:String) : Boolean;
// Renvoi la coordonnée absolue de la souris en X par rapport au coin gauche du synoptique
Function **Xmouse** : Integer;
// Renvoi la coordonnée absolue de la souris en Y par rapport au coin gauche du synoptique
Function **Ymouse** :Integer;
Function **CloseForm**(NomEcran : String) : Boolean ; *// ** Ferme le synoptique spécifié*

Function **FormGetbounds**(FormName : String; **Var** Left,Top,Width,Height : Integer): Boolean;
*//** Renvoie les dimensions d'un synoptique. Left,Top sont les coordonnées du coin gauche haut, Width pour la largeur et Height pour la hauteur. Retourne true si le synoptique existe*
Function **FormSetbounds**(FormName : String; Left,Top,NewWidth,NewHeight : Integer): Boolean;
// Fixe les nouvelles dimensions d'un synoptique (nom contenu dans formname) . Left,Top sont les *coordonnées du coin gauche haut, Width pour la largeur et Height pour la hauteur.*
Si NewWidth ou NewHeight sont égale à 0 la largeur ou la la hauteur n'est pas modifiée.

Instructions concernant les Bitmaps

// Anime un Bitmap en Bouton poussoir simple avec 2 Bitmaps
Function **ButtonOwner** (BmpUp,BmpDown:String) : Boolean;

// Anime un bitmap en bouton à accrochage avec 2 bitmaps
Function **ToggleOwner**(BmpUp,BmpDown:String):

// Fixe le message affiché sur un bouton ou sur une figure texte
Procedure **SetTextMessage** (BmpFile:String);
Procedure **StretchBmp**(State:LongInt); *// Fixe l'attribut d'affichage d'un bitmap.*
Procedure **SetHandleBMP** (Hdle : Integer) ; *// Fixe par programme le Handle d'une figure Image (Bitmap)*

// Renvoie le handle et les attributs d'un Bitmap spécifié
Function **OpenBMP**(BmpName : String ; Var Width : Integer ; Var Height : Integer ; Var Palette : Integer) : Integer;

Instructions concernant les Menus

Function **GetMenuSt** (id : Word) :String; *// Renvoi l'état de l'item demandé.' (coché ou non)*
Variable **MenuId : Integer;** *// Renvoi l'identification de l'item du menu pointé.*
Function **MenuStr**:String; *// Renvoi la chaîne de l'item du menu pointé.*

Procedure **SetMenuString** (Id : Word; Value : String); *// Fixe la chaîne de l'item demandé*
// Fixe la chaîne de la rubrique demandée dans le menu principal
Procedure **SetPopupString** (Id : Word; Value : String);

// Fixe la chaîne de la rubrique demandée dans le menu PopUp (activé par clic droit souris)
Procedure **SetPopupPString** (Id : Word; Value : String);

Procedure **ToggleMenu** (IDitem : Word); *// Bascule l'état coché d'un élément du menu*
Procedure **ValidMenuItem**(Ident:Word; State:Boolean); *// Active ou désactive un item du menu*
Function **GetMenuSt** (id : Word) :String; *// Renvoi l'état de l'item demandé.' (coché ou non)*
Procedure **SetMenuBitmap** (id : Word ; NomBmp:String); *// Fixe un petit Bitmap 16x16 devant un Item de Menu*

Autres et spécifiques

Function **GetTrackBar** : Integer ; *// Renvoi la valeur d'un trackbar*
Procedure **SetTrackBar** (Value : Integer ; *// Fixe la valeur dans un trackBar.*

Procedure **SetTrackEND** (Position : Integer) ; *// Fixe Position d'évolution mini et maxi du TrackBar*
Procedure **SetTrackSTART** (Position : Integer) ; .

Function **GetUPDown** : Integer *// Renvoie la valeur du composant*
Procedure **SetUPDown** (Value : integer) ; *// Fixe la valeur du Composant*

Afficher un schéma à contact dans une fenêtre Window

// Affecte un nom de fichier ladder à la fenêtre Window
Procedure **WinLadder** (LadderName : String) ; ***// ** Ne pas faire au tour de cycle ! ! ! ! *****
Procedure **WinLadderRefresh** ; *// Anime en visu-Dynamique le ladder dans la fenêtre Window*
Procedure **WinLadderZoomIn** ; *// Effectue un zoom en réduction du ladder affiché*
Procedure **WinLadderZoomOut** ; *// Effectue un zoom en agrandissement du ladder affiché*
Procedure **WinLadderClear** ; *// Efface la visualisation du fichier ladder en cours*
☞ **Note** : Utilisez les composants (sécurisés) de la boite à outils

Exemple de programmation derrière une fenêtre Window.

```
If Start_Visu Then WinLadder('C:\MonProj\LAD1.LAD') ; // ** Charge une fois *
WinLadderRefresh ;      // ** Rafraîchi tout le temps *
If ZI Then WinLadderZoomIn Else     // ** test si Zoom demandé *
If ZO Then WinLadderZoomOut ;
```

Afficher un Grafcet dans une fenêtre Window

*// ** Affecte un nom de fichier Grafcet à la fenêtre Window ***
Procedure **WinGrafcet** (LadderName : String) ; ***// ** Ne pas faire au tour de cycle ! ! ! ! *****

Procedure **WinGrafcetRefresh** ; *// Anime en visu-Dynamique le Grafcet dans la fenêtre Window*
Procedure **WinGrafcetZoomIn** ; *// Effectue un zoom en réduction du Grafcet affiché*
Procedure **WinGrafcetZoomOut** ; *// Effectue un zoom en agrandissement du Grafcet affiché*
Procedure **WinGrafcetClear** ; *// Efface la visualisation du fichier Grafcet en cours*

☞ **Note** : Utilisez les composants (sécurisés) de la boite à outils

Manipuler les coordonnées d'un synoptique

Function **GetClientOrigin** (Var Xo,Yo : Integer) : **Integer**;
Retourne la corrdonnée en haut (Yo) à gauche (Xo) du synoptique courant.

Function **GetFormHandle** (NomSyno : String) : **Integer**;
Retourne le handle d'un synoptique

Function **FormSetbounds** (NomSyno : String; X,Y,Nouvlarg,NouvHaut : Integer): boolean;
Nouvlarg : Nouvelle largeur du synotique (en pixels)
Nouvhaut : Nouvelle Hauteur du synotique (en pixels)

Exemple de positionnement d'un synoptique sous un composant :
Uses Graphics, Windows;

```
 If Button Then
 Begin
   Getclientorigin(Xo,Yo); //** Retourne coin haut/gauche du composant
   PtX:=Xo;
   PtY:=Yo+(GetYfin - GetYorg);      //*** Coin bas/droite
   W:=GetSystemMetrics(SM_CXSCREEN); //** résolution Ecran (largeur)
   H:=GetSystemMetrics(SM_CYSCREEN); //** résolution Ecran (Hauteur)
   FormGetBounds('Mykeyboard',BX,BY,BW,BH); //** recupère les coor du syno
   IF (Pty+BH)>H then Pty:=Pty-BH-(GetYfin-GetYorg);
   IF (Ptx)<0 Then Ptx:=0;
   IF (Pty)<0 Then Pty:=0;
   //** positionne le syno sous le composant avec nouvelles coordonnées
   FormSetbounds('Mykeyboard',Pt.X,Pt.Y);
End;
```

Variables Système Visual I/O

DRAGFROM : String ; // Figure à l'origine du Drag & Drop
DBCLICK : Boolean ; // Double click effectué dans la figure
LBUTTONUP : Boolean ; // Relâchement du bouton gauche de la souris
LBUTTONDOWN: Boolean ; // Un clic du bouton gauche de la souris à été effectué
RBUTTONUP : Boolean ; // Relâchement du bouton droit de la souris
RBUTTONDOWN: Boolean ; // Un clic du bouton droit de la souris à été effectué
BT_500 : Byte ; // Passe à 1 tous les 500 ms
BT100 : Byte ; // Passe à 1 tous les 100 ms
CDATE : Real ; // Date en cours codée
CTIME : Real ; // Heure en cours codée
SCDATE : String ; // Date courante
SCTIME : String ; // Heure courante
CYEAR : Word ; // Année courante
CMONTH : Integer ; // Mois courant
CDAY : Integer ; // Jour courant
CHOUR : Integer ; // Heure courante
CMIN : Integer ; // Minute courante
CSEC : Integer ; // Seconde courante
CHSEC : Integer ; // Dixième de seconde courante
START_VISU : Boolean; // Passe à True lors de la première apparition d'un synoptique
END_VISU : Boolean; // Passe à True lors de la fermeture de l'application
WINAPPARITION: Boolean ; // Passe à True lors de chaque apparition du Synop.
WINDISPARITION: Boolean ; // Passe à True lors de chaque disparition du Synop.
MENUID : Word ; // Identificateur du menu cliqué
XMOUSE : Integer ; // Coordonnée courante de la souris en X}
YMOUSE ; Integer ; // Coordonnée courante de la souris en Y}
XPRINTPAGE : Integer ; // Nombre de Pixel en X de la page sur imprimante courante
YPRINTPAGE : Integer ; // Nombre de Pixel en Y de la page sur imprimante courante
ALTACTIVED : Boolean ; // Touche ALT Enfoncée
SHIFTACTIVED: Boolean ; // Une des Touches SHIFT Enfoncée
CTRLACTIVED: Boolean ; // Touche CTRL Enfoncée
SCANFORMS : Integer ; // Temps de scrutation des Synoptiques. **10 ms par défaut** Mini = 1ms
0=Max 10=10 Ms Etc.. Dispatch du temps CPU
PAINTSTRUCT: Integer ; // pointer sur structure pour courbe voir chapitre

EVENTAPP : Boolean; // Indique si un clic souris, clavier à été effectué sur le synoptique
DOTRANSLATE: Boolean; // Autorise ou non la traduction des textes
RUNUSERPROG: Pointer; // Integer pointer sur procédure qui sera appelée par les système cycliquement.
CALLBACKPRINTALARM : Pointer; // Integer pointer sur Procédure qui sera appelée le gestionnaire d'alarmes.
FORMMSGS : Pointer; // Pointeur sur procédure perso pour réception des messages windows.

Variables systèmes en gras : Nouvelles variables dans version 7 de Visual I/O.

GLOBVARSIZE : Integer; // Renvoie la taille de vos variable globales
BASEGLOBVARS : Integer; // Renvoie l'adresse de départ en mémoire de vos variables globales

Nouvelles variables système pour utilisation dans applis sans déclaration de variables globales

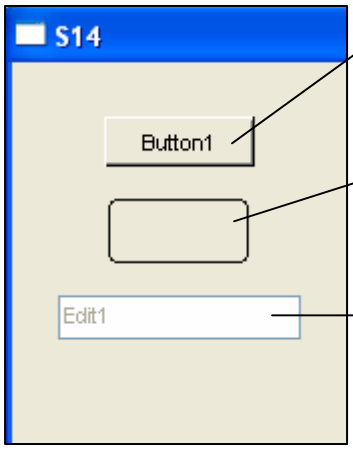
```
INT_Sys1  to INT_Sys10   : Integer;
REAL_Sys1 to REAL_Sys10  : Real;
BOOL_Sys1 to BOOL_Sys10  : Boolean;
STR_Sys1  to STR_Sys10   : String;
```

Différentes zones de l'éditeur de programmes des synoptiques.

Liste des librairies
nécessaire à vos programmes.
Ici les procédures, fonctions et constantes de la librairie Windows sont utilisables dans le programme du synoptique.

Déclaration des variables
locales au synoptique et communes à toutes les figures.
ICI 2 variables I et J et une variable R.

Une **fonction utilisable et commune** à tous les objets graphiques.




```
{** 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;
```

Les différentes portion de code associées à chaque tracé graphique.

Une instruction de coloriage interne **SETFILLCOLOR** pourra colorier l'intérieur du Bouton si celle-ci est utilisée entre **Begin** et **End** de **SObject Procedure Button1**; ou colorier l'intérieur du rectangle ou de l'Editbox si celle-ci est écrite dans le bloc programme associé à l'élément.

Le programme de la figure EDIT1 peut utiliser alors la variable R ainsi que la procédure CalculTTC. Getvalue retourne la valeur numérique frappée dans la zone de saisie Edit1

A partir de la version 7 de Visual I/O, les variables locales à un synoptique peuvent être utilisées dans les experts.

Avant cette version 7 les experts ne pouvaient utiliser que des variables globales.

L'utilisation des variables locales dans les experts, permet de créer des synoptiques relogeables sans soucis dans d'autres applications.

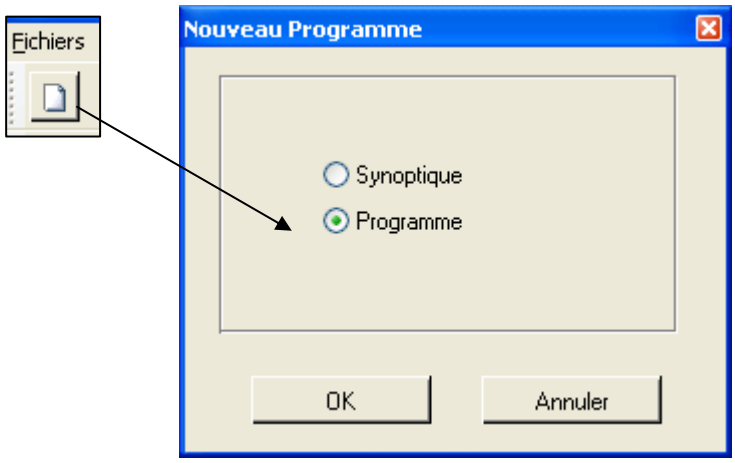
Le programme associé au synoptique se trouve dans le fichier de nom du synoptique plus _AUTO.VIO.

Exemple :

Synoptique	S12.SYN
Source Programme	S12_AUTO.VIO
Programme compilé	S12_AUTO.VPU

Création de librairies VPU ou de programmes cycliques

Visual I/O permet de créer des librairies contenant des procédures ou des fonctions que vous utiliserez dans vos programmes. Les librairies sont aussi appelées **Unités ou plus communément VPU**.



Cliquez sur nouveau. Une fenêtre apparaît. Cochez le radio bouton **Programme** puis le bouton OK.
Un éditeur de texte apparaît permettant de créer des programmes ou des librairies.

Ecriture des programmes cycliques en langage Pascal

Il est possible sous Visual I/O d'écrire un programme cyclique (VPU) qui sera chargé dans les taches prioritaires (Menu utilitaire/constitution application).

L'éditeur permet l'écriture de librairies ainsi que des programmes cycliques. Dans le cas d'un programme effacez la première ligne contenant le mot réservé UNIT ainsi que la ligne contenant le mot Interface.

Un programme simple peut ressembler à :

```
Uses MVarglob;  //  Utilise la Vpu des variables globales Mvarglob *
Begin           // *** Begin du corps principal **
  D0:=D0+1;
End.            // *** End du corps principal **
```

Autre exemple :

```
Uses MVarglob;  //  Utilise les variables globales Mvarglob *
//**** Procedure addition ***********
Procedure CalculPlus( Var XWrd : Integer);
Begin
  XWrd:=XWrd+1;
End;
//*** Procedure de conversion degrees vers radiants ******
Function DegreRad ( Xdegre : Real) : Real;
Begin
  Result:= Xdegre *(3.1415926535897932385 /180);
End;

//*** Corps du programme cyclique *******
Begin           // *** Begin du corps principal **
  D0:=D0+1;
  Calculplus(D0); //*** équivalent à D0:=D0+1;
  Angle:=DegreRad(Mesure); //** Conversion
  Temperature:=Scale(ValPLC,0,32767,0,100); //** Mise à l'échelle
End.            // *** End du corps principal **
```

Note : Comme il s'agit ici d'un programme, les procedures et fonctions utilisées ici, ne sont accessibles que par ce programme. Si vous voulez utiliser ces procedures et ces fonctions dans d'autres programmes il faut les incorporer dans une librairie, puis ajouter le nom de celle-ci dans la clause Uses derrière MVarglob. (*'Uses MVarglob,xxx;'*).

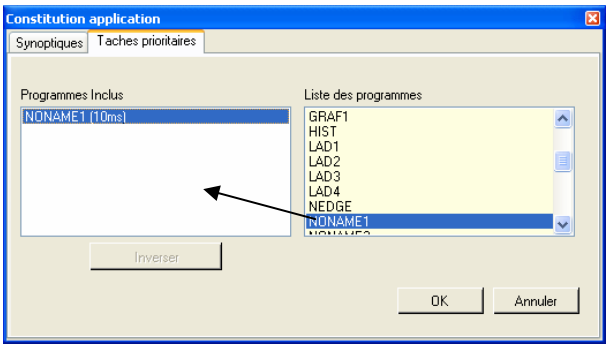
La clause USES

USES est un mot réservé du langage, indiquant au programme qu'il peut utiliser des variables, des procédures et des fonctions localisées dans les librairies nommés.

Exemple USES Mvarglob, Calcul, Personnel; Indique au programme qu'il peut utiliser tout ce qui est fournit par les librairies Mvarglob, Calcul et Personnel.VPU.

Le corps du programme

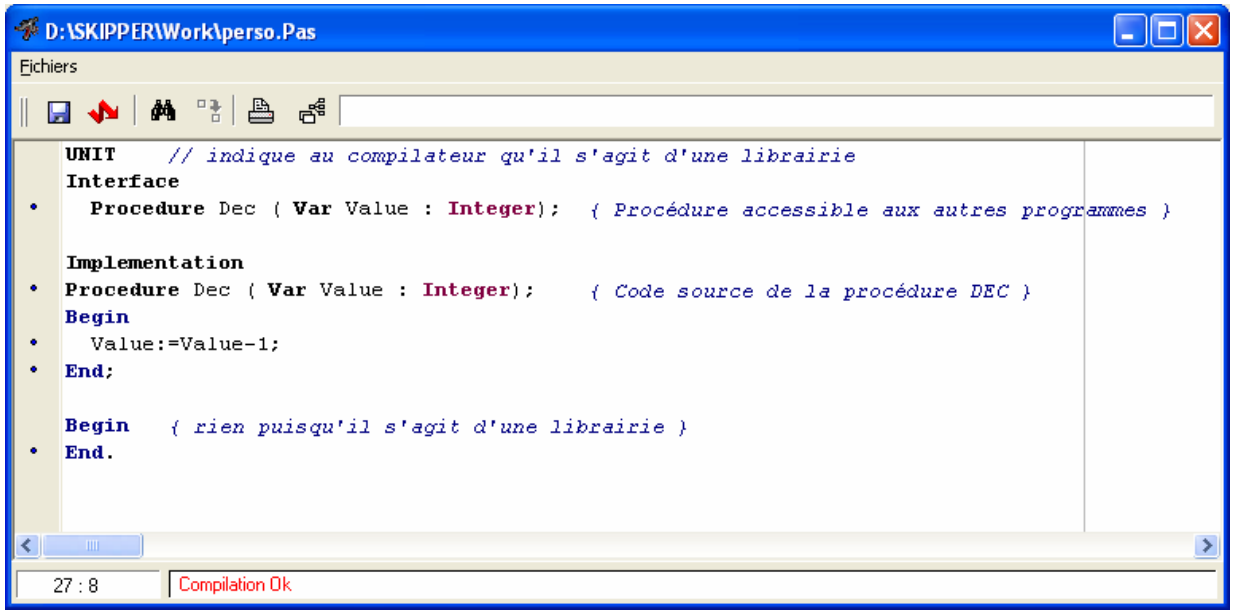
Ce corps de programme contient les instructions qui doivent êtres appelées cycliquement par la tache prioritaire du programme généré par visual I/O.
Ce corps commence par un **BEGIN** et fini par un **END.** (suivit d'un point et non d'une point virgule).



Compilez et Chargez le programme écrit à l'aide du menu Utilitaires/Constitution Application et Taches prioritaires.

Ecriture de librairies VPU

Le principe décrit avant est identique hormis qu'une librairie ne contient pas de programme cyclique. Une librairie est passive et met au service des différents programmes externes des variables, procédures et des fonctions.



Imaginons une librairie contenant une procédure DEC. Celle-ci à pour but de décrémenter une valeur numérique.

Examinons ce code :

```
UNIT        //  indique au compilateur qu'il s'agit d'une librairie
Interface
 Procedure Dec ( Var Value : Integer);    // Procédure accessible aux autres programmes

Implementation

Procedure Dec ( Var Value : Integer);   // Code source de la procédure DEC
Begin
```

```
  Value:=Value-1;
End;

Begin //  rien puisqu'il s'agit d'une librairie
End.
```

La clause UNIT

Indique au compilateur qu'il s'agit d'une **librairie** et non pas d'un programme donc rien à exécuter cycliquement dans le corps principal contenu entre BEGIN et END.

En **INTERFACE** nous déclarons le prototype (nom de la procédure ainsi que ses différents paramètres).
En **IMPLEMENTATION** nous retrouvons la procédure Dec complète avec son code source.
Entre BEGIN et END. du corps principal aucune instruction puisqu'il s'agit d'une librairie.

La procédure DEC : Nous nous apercevons que Value est passée en paramètre avec le mot **VAR** devant. Quand un paramètre est précédé du mot VAR, cela signifie qu'il peut être modifié par le programme de la procédure ou de la fonction le recevant.
En fait le paramètre reçu par la procédure n'est pas le contenu de Value mais son adresse. Le programme DEC décrémente le contenu de la variable à l'adresse passée.

Initialisation dans une librairie

Puisqu'il s'agit d'une librairie aucune ligne de programme n'était frappée entre BEGIN et END (corps principal). Cependant il est possible d'écrire du code source à cet endroit. Celui-ci sera alors exécuté une seule fois au lancement de l'application.
Cette fonctionnalité pourra être utilisée pour initialiser des variables ou des équipements avant le démarrage effectif de l'application.

Examinons une variante du code précédent :

```
UNIT          // indique au compilateur qu'il s'agit d'une librairie
Interface
  Procedure Dec ( Var Value : Integer);   // Procédure accessible aux autres programmes

Implementation

Procedure Dec ( Var Value : Integer);     // Code source de la procédure DEC
Begin
  Value:=Value-1;
End;

Procedure Initial;      // procédure interne à la librairie
Begin
   …..
 End;

Begin  // Initialisation de la librairie au démarrage de l'application
  Initial;   // l'appel à initial est fait qu'une seule fois
End.
```

Passage de paramètres par valeurs et par adresse

Lorsque que vous créez des procédures et des fonctions, vous allez devoir passer des paramètres à ces procédures et fonctions. Deux types de passage de paramètres sont autorisés.

Passage par valeur

```
Ex : Procedure Test ( Value : Integer);
Begin
End;

Begin
  I:=10;
  Test(I); // ** passage par valeur à la procédure Test
           // ici I sera toujours égal à 10 quel que soit l'action dans Test
End.
```

Dans cette procédure Value correspondant à I ne pourra être modifié par la procédure. On pourra écrire :
If Value = 10 Then … ou Compteur:=Value;

La commande Value:=100 n'aura aucune répercussion sur la variable passée à l'origine à la procédure (ici i). Le fait d'affecter value d'une valeur dans la procédure, modifie la valeur locale de value mais non la valeur originale. Dans la procédure Test une recopie locale de I est faite.

Passage par adresse

```
Ex : Procedure Test( Var Value : Integer);
Begin
   Value:=Value*2;   // *** Modifie en fait l'état de i *
End;
Begin
   I:=10;
  Test(I);   // ** passage par adresse à la procédure Test *
        // ici I n'est pas forcément égal à 10 cela dépend de la procédure Test
End
```

Reprenons l'exemple de la procédure Inc qui permet d'incrémenter une variable de 1.

On pourra écrire :

```
Procedure INC ( Var Value : Integer);
Begin
  Value:=Value+1;
End;

Begin
  Inc(I); // ** incrémente I de 1
End.
```

Ou écrire une fonction qui effectuera la même action :

```
Function INC ( Value : Integer) : Integer;
Begin
  Result:=Value+1;
End;

Begin
  I:=inc(I);
End.
```

Autre exemple :

```
Function Inc(Value : Integer) : Integer;
Begin
  Result:=Value+1;
End;

Begin
  R:=1.2;
  R:=Inc(R);   // ** passage d'un réel à la fonction et résultat dans un réel
End.
```

La fonction INC est normalement prévue pour recevoir et renvoyer un Integer.
Dans ce cas le compilateur transforme R qui est un réel en un Integer pour s'adapter à la fonction et transforme aussi le résultat renvoyé sous forme d'un Integer en réel.

Passage d'une variable sans type par adresse

On peut aussi écrire :

```
Procedure Inc ( Var Value);
Begin
   Integer(Value):=Integer(Value)+1;
End;
```

Ici la procédure INC reçoit une variable de type inconnu et ceci par adresse. Il faut que la procédure transtype la variable dans un type de son choix. La procédure INC décide dans cette exemple que value est de type Integer.

Les instructions système de Visual I/O par catégories

Les grilles de données

Visual I/O un ensemble des fonctions permettant de gérer les grilles.
Une grille regroupe un ensemble de données (valeurs ou autres) dans un tableau. Visual I/O met à disposition du programmeur un ensemble de fonctions permettant de manipuler par programme les données de la grille.
Une grille peut contenir du texte, des bitmaps et des contrôles Windows (Listebox, ComboBox, Cases à cocher), des lignes de différentes couleurs.
Pour la création d'une grille de données, 2 possibilités

Soit un utilisant le composant de la boite à Outils (Grille de données).
Soit par programme derrière une fenêtre (Window).

Pour les différentes fonctionnalités se rapporter au chapitre grille de données

Cas de plusieurs grilles dans un synoptique

Une seule grille est présente dans un synoptique :
Si une seule grille est présente dans un synoptique, toutes les fonctions se rapportant à une grille sont destinées à celle-ci.

Plusieurs grilles présentes dans un synoptique :
Il est possible d'avoir plusieurs grilles dans un même synoptique. Pour adresser alors une commande à une grille précise, il faut effectuer la commande Go2(N) signifiant Commuter sur la figure de nom N qui a créer la grille à l'aide de cette procédure, le système sait alors que l'on adresse cette grille et non une autre.

Fonctionnalités orientées Initialisation

(Assigne la grille dans la fenêtre et initialise la grille).
Procedure **CustomGrid** (Col,Rows, FixedCol,FixedRow : Integer); *// * Rangée,Colonnes*

// Fixe les différentes options de la grille
Procedure **GSETOPTIONS** (Options : Integer) ; *// combinaisons d'options*

FixedHorzLine	Trace les lignes Horizontales dans les Zones cellules Fixes.
FixedVertLine:	Trace les lignes Verticales dans les Zones cellules Fixes.
HorzLine:	Trace les lignes horizontales entre les Cellules.
VertLine:	Trace les lignes Verticales entre les Cellules.
RangeSelect:	Permet aux cellules d'êtres Sélectionnées.
DrawFocusSelected:	Met en Surbrillance la Cellule Sélectionnée.
RowSizing:	Permet aux Rangées d'êtres individuellement retaillées en Hauteur.
ColSizing:	Permet aux Colonnes d'êtres individuellement retaillées en Largeur.
RowMoving:	Permet aux Rangées d'être déplacées avec la Souris.
ColMoving:	Permet aux Colonnes d'être déplacées avec la Souris.
HideEditing:	Cache le Contour de la Cellule Sélectionnée.
DisableEdit:	Interdit la Saisie pour modification des Cellules.
Tabs:	Autorise le "tabbing" entre colonnes.

AlignCenter, AlignRight Aligne les textes dans les cellules au centre ou à droite.

Valeurs Numériques associées

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;	AlignCentre	= 16384;	AligneDroite = 32768;

Exemple : GsetOptions(FixedHorzLine Or FixedVertLine) ; *// Equivalent GSetOptions(3);*
(Affecte une commande à une cellule).
Procedure **GCommand** (Tipe : String; Col, Row : Integer);

Tipe : 'COLOR' ou 'BITMAP' ou 'CONTROL'.
Col : Numéro de colonne de la cellule affectée par la commande.
Row : Numéro de rangée de la cellule affectée par la commande.

☞ **Note** : Voir chapitre sur les grilles

*(** Permet de fixer la largeur d'une colonne).*
Procedure **GSetColWidth** (C : Integer; V : Integer);

(Assigne un contrôle Windows à une cellule).
Procedure **GsetControl** (ACol, ARow: integer; Value : string);

*(** Permet de fixer la couleur de fond des colonnes et rangées fixes).*
Procedure **GSetFixedColor** (C : Integer);

*(** Permet de fixer le nombre de colonnes fixes).*
Procedure **GSetFixedCols** (C : Integer);

*(** Permet de fixer le nombre de rangées fixes).*
Procedure **GSetFixedRows** (R : Integer);

*(** Fixe la hauteur d'une rangée).*
Procedure **GSetRowHeight** (R : Integer; V : Integer);

*(** Fixe la couleur d'écriture des colonnes et rangées fixes).*
Procedure **GSetTextFixedColor** (Color : Integer);

(Autorise l'entrée en saisie par Double Click dans toutes les cellules).
Procedure **GSetDoDbClick** (Enable : Boolean);

Fonctionnalités orientées navigation

(Indique si une cellule a changé de contenu).
Function **GCellChange** : Boolean;

(Teste si l'utilisateur a changé de cellule).
Function **GDeplaceFocus** : Boolean;

Fonctionnalités orientées contrôle

(Test un doubleClic dans une cellule de la grille).
Function **GDbClick** : Boolean;

(Autorise l'a saisie dans les cellules).
Procedure **GValidEdit**;

(Dévalide la possibilité de saisie dans les cellules).
Procedure **GDevalidEdit**;

(Valide ou dévalide l'affichage dans une grille).
Procedure **GEnableDisplay**(On_OFF : Boolean);

(Colorie une cellule).
Procedure **GSetCellColor**(ACol, ARow: Integer; Couleur : Integer);

*(** Permet de se positionner à une colonne).*
Procedure **GSetCol**(V : Integer);

(Après Initialisation, fixe le nombre de colonne).
Procedure **GSetColCount**(V : Integer);

*(** Permet de positionner à gauche une colonne / scrolling).*
Procedure **GSetLeftCol** (C : Integer);

*(** Permet de positionner en haut une rangée / scrolling).*
Procedure **GSetTopRow** (Row : Integer);

*(** Permet de se positionner à une rangée donnée).*
Procedure **GSetRow** (V : Integer);

(Après Initialisation, fixe le nombre de rangées).
Procedure **GSetRowCount** (V : Integer);

(Fixe la couleur du texte d'une cellule).
Procedure **GsetTextColor** (Colum, Row : Integer ; Color : Integer) ;

Fonctionnalité orientée Information

(Récupère le numéro de ligne de colonne de la cellule pointée par la souris).
Procedure **FlyingCells** (Var ACol, ARow: Integer);

*(** Retourne la couleur d'une cellule).*
Function **GGetCellColor**(Var ACol, ARow: Integer) : Integer;

*(** Retourne la largeur d'une colonne).*
Function **GGetColWidth**(Index: Integer) : Integer;

(Récupère le numéro de ligne de colonne de la cellule courante).
Procedure **GGetCurrent** (**Var** ACol, ARow: Integer);

*(** Retourne la couleur de la cellule spécifiée).*
Function **GGetTextColor** (Colum, Row : Integer) : Integer ;

*(** Retourne la hauteur d'une rangée).*
Function **GGetRowHeight**(Index: Integer) : Integer;

*(** Retourne la colonne qui à changée d'état) .*
Function **GGetColChanged** : Integer;

*(** Retourne la rangée qui à changée d'état) .*
Function **GGetRowChanged** : Integer;

*// ** Retourne le nombre de rangées de la grille **
Function **GGetRowCount** : Integer;

*// ** Retourne le nombre de Colonnes de la grille **
Function **GGetCowCount** : Integer;

Fonctionnalité orientée Lecture/Ecriture

(Efface le contenu de la grille).
Procedure **GClearGrid**;

(Récupère le texte dans une cellule).
Function **GGetText**(ACol, ARow: Integer) : string;

(Ecrit le texte dans une cellule).
Procedure **GSetText** (ACol, ARow: Integer; Value : string);

(Ecrit le texte dans la cellule courante).
Procedure **GSetTextCurrent** (Value : string);

(Renvoie le nombre de colonnes de la grille).
Function **GGetColCount** : Integer;

(Renvoie le nombre de rangées de la grille).
Function **GGetRowCount** : Integer;

// Autorise ou non la saisie de c&ractères unicode dans la grille
Procedure **GsetUnicode** (Value : Boolean) ;

// Autorise ou non l'entrée en saisie dans les cellules de la grille par double click
Procedure **SetDoDBclick** (Value : Boolean) ;

Gestion du Multi-Média

Visual I/O permet de mettre en œuvre des commandes multimédia très simplement. Des fonctions pour jouer des sons (*.Wav ou *.Mid) mais aussi des animations vidéo de type *.AVI.

Pour jouer un fichier média il faut définir un cadre ou sera affiché le média (AVI par exemple). Toute figure rectangle peut être utilisée (Rectangle, BMP, Support etc..).
Ce cadre permettra de renseigner le 'Média-Player' de la délimitation d'affichage dont il dispose.
L'Initialisation du player (InitPlayer) est en général demandé avec l'argument Etirer=vrai.

Les 3 instructions principales sont :

InitPlayer : Initialisation de la zone multimédia.
PlayMedia : Jouer un Média.
NewMedia : Indiquer un nouveau fichier multimédia à jouer.

Les autres commandes sont des commandes annexes pour mettre en pause au début etc..

Exemple de programme simple pour jouer un fichier AVI.

Dans un rectangle:

```
InitPlayer(' ',True);
NewMedia('Fanfar.Avi');
PlayMedia;
```

Liste des Instructions

// Recul d'un pas dans le fichier Média courant.}
Procedure **BackMedia**;

// Renvoi la longueur du Média courant.}
Function **GetLengthMedia** : LongInt;

// Renvoi la position en cours dans le Média courant.}
Function **GetPositionMedia** : LongInt;

// Initialise une zone rectangulaire servant de cadre au MédiaPlayer.}
Procedure **InitPlayer** (Name : String; Stretch : Boolean);

// Indique si le Média courant à été joué entièrement.}
Function **MediaAtEnd** : Boolean;

// Indique si le Média courant est à été joué et est au début.}
Function **MediaAtZero** : Boolean;

// Indique si le Média courant est entrain d'être joué.}
Function **MediaPlaying** : Boolean;

// Assigne un nouveau nom de fichier multimédia à jouer.}
Procedure **NewMedia** (Nom : String);

// Passe à l'enregistrement suivant sur un CD-ROM.}
Procedure **NextTrack**;

// Met en pause le Média en cours.}
Procedure **PauseMedia**;

// Met en pause le Média en cours.}
Procedure **PauseOnlyMedia**;

// Démarre une séquence multimédia.}
Procedure **PlayMedia**;

// Passe à l'enregistrement précédent sur un CD-ROM.}
Procedure **PreviousTrack**;

// Remet en marche le Média courant arrêté.}
Procedure **ResumeMedia**;

// Met le média courant au tout début.}
Procedure **RewindMedia**;

// Avance le média courant d'un pas.}
Procedure **StepMedia**;

// Arrête le média courant.}
Procedure **StopMedia**;

// Joue un fichier .WAV ou .Mid
Procedure **PlaySound** (WavFileName : String) ;

// Emet un bip sur le haut-parleur du PC}
Procedure **Beep** ;

Gestion DDE

Outre ses composants DDE, Visual I/O permet de piloter une liaison DDE par programme.
Attention par programme, un seul canal DDE ne peut être ouvert.

// Ouvre le canal DDE
Procedure **OpenDDE** (Server : String) ;

// Ferme le Canal DDE
Procedure **CloseDDE** ;

// Récupère une donnée dans le serveur DDE ouvert avec OpenDDE
Function **DDEGetData** (Topic, Item : String) : String ;

// Envoie une donnée dans le serveur DDE ouvert avec OpenDDE
Function **DDESendData** (Topic, Item, Value : String) : String ;

// Envoie une Macro-commande dans le serveur DDE ouvert avec OpenDDE
Function **DDESendMacro** (Macro : String) : String ;

☞ **Note** Voir aussi composant DDE.

```
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               {** Retrieves 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 ;
```

Gestion de la liaison série

Visual I/O grace à son Unité Serial.VPU permet de gérer les liaisons série COM1 à COM8. Cette Vpu est livrée avec le source.
☞ **Note** Voir les composants Serial Com Receiver et serial coms transmitter de l'Onglet liaison série de la boite à outils

Liste des Instructions de Serial.Vpu

// Ouvre un canal série
Function **OpenCom** (ComPort,Baudrate,Parity,Bits,Stop:String): Boolean;

// Test si le Canal série spécifié est déjà ouvert
Function **GetComOpen** (Comport : Integer) : Boolean;

// Ferme un canal série}
Procedure **DisableCom** (ComPort : Byte);

// Ferme tous les canaux série}
Procedure **DisableAllComs**;

// Vide le buffer de réception d'un canal série
Procedure **ClearCom** (ComPort : Byte);

// Test si un caractère est présent dans le buffer de réception
Function **V24DataOk** (ComPort : Byte) : Boolean;

// Test le nombre de caractères dans le buffer d'émission ou de réception
Function **ComBufferLeft**(ComPort : Byte; IO:Char) : Word; // IO='I' ou 'O' I=Input O=Output}

// Récupère un caractère dans le buffer de réception
Function **GetByte** (ComPort : Byte) : Byte;

// Envoie un caractère dans le buffer d'émission
Procedure **SendByte** (ComPort,Data : Byte);

// Envoie une zone d'octêts dans le buffer d'émission
Procedure **ComWriteBuf** (ComPort:Byte; Var PointerTable; NB : Integer);

// Envoie une chaîne de caractères dans le buffer d'émission
Procedure **ComWrite** (ComPort :Byte; Value : String);

// Test si le buffer est prêt à émettre
Function **ReadComOk**(ComPort:Byte) : Boolean;

// Positionne un flag interne indiquant que la communication est déjà prise
Procedure **SetComBusy** (Comport: Byte);

// Fait tomber le flag interne indiquant ainsi que la communication est libre
Procedure **ResetComBusy** (Comport: Byte);

// Affecte le signal DTR à 1 ou 0 sur le port spécifié
Procedure **Set_DTR** (ComPort: Byte; OnOff: boolean);

// Affecte le signal RTS à 1 ou 0 sur le port spécifié
Procedure **Set_RTS** (ComPort:Byte; OnOff: boolean);

// Teste l'état du CTS sur le port spécifié
Procedure **Read_CTS** (ComPort: Byte; var cts_value: byte) : Boolean;

☞ **Note** : Pour exemples de programmes voiur chapitre Instructions pour le pilotage de la liaison série.

Gestion de l'imprimante en direct

Visual I/O permet d'imprimer des chaînes de caractères plus simplement qu'avec les outils traditionnels de Windows. Cette librairie donne accès à un jeu d'instructions simple permettant de remplir un buffer d'impression destiné à votre imprimante.
Le buffer d'impression est initialisé par la commande **InitPrinter**, et l'impression est déclenchée en fin de traitement par la commande **FPrint**.
L'impression travaille par rapport à un format de papier dépendant de l'imprimante connectée par défaut.
Pour travailler avec une autre imprimante ou éventuellement dans un autre format (Paysage par exemple) utilisez le gestionnaire d'impression de Windows pour paramètrer le contexte voulu.
Vous pouvez connaître avant ou pendant l'impression le nombre de pixels utiles sur votre imprimante en lisant les variables **XPrintPage** : Nombre de pixels en X imprimables (Largeur utilisable).
YPrintPage : Nombre de pixels en Y imprimables (Hauteur Utilisable).

Procedure **SETPRINTERFORMAT** (Portrait : Boolean) ;
Permet de passer votre imprimante courante en portrait ou paysage.
Pour positionner la tête d'impression à un endroit donné utilisez la commande **SetMarge**.
Celle-ci fixe la marge gauche.

Liste des Instructions

// Initialise le Buffer d'imprimante interne de Visual Pascal}
Procedure **InitPrinter**;

// Effectue un saut de page.}
Procedure **FormFeed**;

// Envoie le contenu du buffer interne d'impression sur l'imprimante.}
Procedure **FPrint**;

// Renvoie le nombre de lignes imprimables dans une fonte donnée.}
Function **GetMaxLines** (WFont : String; WSize : Integer) : Integer;

// Renvoie la hauteur d'un texte dans une Fonte ayant une taille donnée.}
Function **HPrintText** (Text,Font : String; Wsize : Integer) : Integer;

// Renvoie la largeur d'un texte dans une Fonte ayant une taille donnée.}
Function **LPrintText** (Text,Font : String; Wsize : Integer) : Integer;

// Envoie un texte dans le buffer interne d'imprimante.}
Procedure **Print** (S : String);

// Fixe les attributs des caractères d'impression.}
Procedure **PrintAttr** (A : Integer);

// Imprime un Bitmap en position X à la ligne Courante.}
Procedure **PrintBitMap**(X : Integer;N : String);

// Imprime un cercle ayant un rayon donné.
Procedure **PrintCircle** (XC,YC,R : Integer; Epaisseur : Integer);

*// ** Fixe la couleur d'impression pour les caractères suivants envoyés ****
Procedure **PrintColor** (Color : Integer);

// Change la la fonte d'impression.
Procedure **PrintFont** (N : String);

// Imprime une ligne.
Procedure **PrintLine** (Xo,Yo,Xf,Yf : Integer; Epaisseur : Integer);

// Envoie un texte suivi de Cr et Lf dans le buffer interne d'imprimante.
Procedure **PrintLn** (S : String);

// Fixe la taille des caractères d'impression.
Procedure **PrintSize** (S : Integer);

// Fixe une marge gauche d'impression.
Procedure **SetMarge**(X : Integer);
// Permet de positionner le stylo dans la feuille en cours.
Procedure **SetPen** (Y : Integer);
// Renvoi en pixels la largeur d'impression utile de l'imprimante par défaut.
Variable **XPrintPage** : Integer;
// Renvoi en pixels la hauteur d'impression utile de l'imprimante par défaut.
Variable **YPrintPage** : Integer;

Function **GetPrinterIndex** : Integer; *//Renvoi le numéro d'imprimante sélectionné dans Windows*.
Procedure **SetPrinterIndex** (PrinterNum : Integer); *//** Fixe une autre imprimante que celle trouvée par défaut au lancement de votre application.*
☞ **Note** : Voir le composant **PrinterSettings** de l'onglet Files pour le changement de l'imprimante.

PrintAttr (Réglages des attributs d'impression)

Action Fixe les attributs des caractères d'impression.
Syntaxe **Procedure** PrintAttr (**A** : Integer);

Description
Cette procédure permet de fixer les attributs des caractères imprimés après cette commande. Cette procédure peut être appelée plusieurs fois dans un programme afin de changer les attributs des prochains caractères imprimés. Par défaut l'attribut est **Normal**.

A Attributs des caractères imprimés.

Peut être une des valeurs suivantes :

Normal	= 1;	Regular	= 2;
Bold	= 3;	Italic	= 4;
Underline	= 5;	StrikeOut	= 6;

Limites InitPrinter doit avoir été effectué.
Exemple PrintAttr(Bold);

SetPrinterFormat (Portrait : boolean) ; *// ** si Portrait est à true impression en portrait sinon impression en paysage*
Ex : If Start_Visu Then **SetPrinterFormat(**false) *; // *** Imprime après en paysage*

Ouverture d'un fichier puis l'imprimer

```
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 concernant les courbes

Un visionneur de courbes peut être utilisé tel quel avec les différents composants de la boite à outils. Cependant si il est nécessaire de créer un nouveau composant ou simplement faire un tracage particulier il est possible d'utiliser les instructions système de traçage de courbe. Pour créer un nouveau traceur de courbes il faut utiliser la procedure CustomTrend détaillée après. Cette instruction est à utilisée derrière une figure Window.

Le visionneur de courbes peut contenir jusqu'à 10 courbes de différentes couleurs. Les courbes sont affichées selon l'échelle Y.

Un zoom en Y peut être effectué en changeant les propriétés de l'axe Y par les procédures **TSetLRangeY** et **TSetHRangeY**.

Un zoom en X ne peut être fait que par programme, puisque chaque point est enregistré par celui-ci sur l'axe X. L'axe X n'a pas d'échelle il peut contenir des temps ou des valeurs numériques (Format Abaque).

Plusieurs lignes peuvent être affectées aux textes des graduations en X, il suffit d'insérer un <CR>+<LF> dans le texte à afficher.

Une **barre de seuil** Haut et Bas peut être visualisée en permanence par la procédure **TSetLLevel** et **TSetHLevel.**

Une **barre verticale** Gauche et Droite peut être visualisée en permanence par la procédure **TSetBLeft** et **TSetBRight.**

Une fonction renvoi la valeur équivalente du pointeur de la souris lorsque celle-ci est sur la courbe.
Une fonction renvoie le numéro du point ou se trouve le pointeur de la souris.
Une fonction indique aussi si le pointeur de la souris c'est déplacé sur la courbe pouvant ainsi déclencher un calcul quelconque.

Une courbe peut avoir des points de différentes couleurs, il suffit de changer de couleur avant d'exécuter la procédure **TSetPoint**. A partir de ce moment le point est enregistré dans la courbe dans sa position sa valeur et sa couleur.

Classification des instructions

Fonctionnalité: Initialisation

Function **CustomTrend** (NB_Col,NB_Rang,NB_ColFixe,NB_RowFixe : Integer) : Integer;
(assigne un visionneur de courbes dans la fenêtre et initialise celui-ci et retourne son Handle).

Procedure **TSetPencilColor** (N : Integer, Color : Integer);
(crée un stylo permettant de tracer une courbe).

Procedure **TSetLRangeY** (V : real);
(fixe l'échelle basse de Y).

Procedure **TSetHRangeY** (V : real);
(fixe l'échelle haute de Y).

Procedure **TSetLLevel** (V : real);
(fixe l'a barre de seuil bas en Y).

Procedure **TSetHLevel** (V : real);
(fixe l'a barre de seuil haut en Y).

Procedure **TSetLLevelCol**(Color : Integer);
(fixe l'a couleur de la barre de seuil bas en Y).

Procedure **TSetHLevelCol** (Color : Integer);
(fixe l'a couleur de la barre de seuil haut en Y).

Procedure **TSetDecimals** (Decimals : Integer);
(fixe le nombre de décimales sur l'axe Y)

Plus facile
Utilisez les composants
de l'onglet Trends

Procedure **TSetBorderRight** (Border : Integer);
(fixe le nombre de pixels de retrait de la dernière graduation sur l'axe X)

Procedure **TSetBLeftCol**;
(Fixe la couleur de la barre verticale gauche)

Procedure **TSetDIVX** (NDivX : Integer) ;
*// *** Fixe le nombre de division principales l'axe X ****

Procedure **TSetDIVY** (NDivY : Integer) ;
*// *** Fixe le nombre de division principales sur l'axe Y ****

Procedure **TSetBRightCol**;
(Fixe la couleur de la barre verticale droite)

Fonctionnalité: navigation

Procedure **TSetPoint** (Stylo : Integer; X : integer; valeur : real);
(affiche un point en X avec une valeur)

Procedure **TShiftLeft**;
(décale à gauche d'une graduation principale)

Procedure **TShiftRight**;
(décale à droite d'une graduation principale)

Procedure **TSetBLeft**;
(Fixe la position de la barre verticale gauche)

Procedure **TSetBRight**;
(Fixe la position de la barre verticale Droite)

Function **TGetSelection** : Integer ;
(Renvoie la sélection faite à la souris)

Fonctionnalité: contrôle

Procedure **TDisplayGrid** (D : Boolean);
(fait afficher ou disparaître une grille).

Procedure **TEnableDisplay** (On_Off : Boolean);
(Autorise ou non l'affichage des points dans la grille).

Procedure **TSetTextX** (N : Integer, S: String);
(fixe un texte sous une graduation principale en X).

Procedure **TSetCoef** (Pen : Integer; Coef : Real);
(fixe un coefficient multiplicateur pour un stylo donné).

Plus facile
Utilisez les composants
de l'onglet Trends

Procedure **TResetPoints** (Stylo : byte);
(efface tous les points d'une courbe

Procedure **TResetAllPoints**;
(efface tous les points de toutes les courbes)

Procedure **TRegisterProc** (P : Pointer);
(enregistre l'adresse d'une procédure pour modifier le type de tracé)

Function **TGetHandleBmp** : HBitmap;
(renvoi un handle de bitmap sur un visionneur)

Procedure **TrendToReport** (SynoReport : String; ControlName : String; TrendHandle : Integer);
*// *** Affecte un visionneur créé par programme à un rapport **

Fonctionnalité: Informations

Function **TGetInterX**;
(renvoi le nombre de pixels entre 2 graduations principales en X)

Function **TGetNptX**;
(renvoi le nombre de pixels total en abscisse (axe X))

Function **TGetNptY**;
(renvoi le nombre de pixels total en ordonnées (axeY))

Function **TGetGraduX** (N : integer) : String;
(renvoi le texte associé à une graduation principale en X)

Function **TYMouse** : Real;
(renvoi la valeur de la souris en Y)

Function **TPtXMouse** : LongInt;
(renvoi la valeur du point X de la souris)

Function **TMouseMove** : Boolean;
(renvoi si la souris a bougée sur la courbe)

Function **TGetDivX** : Integer ;
(renvoie le nombre de divisions principales en X

Function **TGetDivY** : Integer;
(renvoie le nombre de divisions principales en Y

Function **TGetYMax** : Integer;
(renvoie la valur de l'échelle max de l'axe Y

Function **TGetYMin** : Integer;
(renvoie la valur de l'échelle mini de l'axe Y

Plus facile
Utilisez les composants
de l'onglet Trends

Exemples de programme

```
// **** dans variables locales ****
Uses Windows,Sysplc ;
Var
  Point     : Integer;
  Grille    : Boolean;
  Valeur    : Real;
// **** Dans Window 1 ***************
If Start_Visu Then
Begin
  Customtrend(5,3,2,1,0,300); // déclaration du visionneur}
  TsetPencilColor(1,0);      // couleur noir du stylo n°1}
  Point:=0;
End;
// *** affichage ou non de la grille sur clic souris ***
If ToplBdown Then                    // clic gauche Souris}
Begin
  TDisplayGrid(Grille);       // affichage de la grille}
  Grille:=Not(Grille);        // flipflop sur la grille}
End;
// * affichage de la valeur (aléatoire) toutes les 100 ms *
If BT_100=1 Then              // traçage toutes les 100 ms}
Begin
  Valeur:=Random;            // calcul de Y en valeur aléatoire}
  TSetPoint(1,Point,valeur);  // traçage du point}
  Inc(Point);                 // incrémentation en abscisse}
  If Point>TGetNptx Then      // arrive en bout d'affichage}
  Begin
   TshiftLeft;          // décalage à gauche: courbe glissante}
   Point :=Point -TgetInterX;       // recalcule du 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); // ** Récupère le handle du visionneur
    TSetPencilColor(1,255);
    TSetPoint(1,10,10);
    TSetPoint(1,100,100);
    TSetPencilColor(2,1255);
    TSetPoint(2,20,20);
    TSetPoint(2,200,200);
    TrendToReport('rapport','Window1',THDLE); // Syno Rapport.syn, Figure Window1
  End;
End;

// *** Button1 ************************************
SObject Procedure Button1;
Begin
  If Button Then PrintReport('Rapport');
End;
```

Personnalisation du tracé des points dans les courbes

Les composants vus précédemment font appel aux instructions de Visual I/O pour le management graphique des courbes.
Le principe adopté pour la personnalisation des tracés est le suivant :

- Si aucune adresse CallBack n'a été spécifiée, les courbes sont tracées par des lignes droites entre chaques points fixés par programme (TSetPoint).

- Si une adresse CallBack est spécifiée, le tracé graphique n'est plus fait par le système mais la procédure de CallBack est appelée, le système lui passant une structure contenant des informations sur le point qui doit être tracé.

La Structure StrucPaint

Cette structure sera utilisée avec les instructions graphiques de l'API Windows (GDI.DLL).
Se référer documentations SDK Microsoft.

```
//******** Custom trend ******
  StrucPaint = Record
       PDC : Integer;  // ***** Drawing Context **
       X   : Integer;
       Y   : Integer;
       Color     : LongInt;
       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;
```

```
PDC          : Contexte d'affichage
X ,Y         : Coordonnées graphiques du point à afficher.
Color        : Couleur du point à afficher.
OrigineX,
OrigineY     : Coordonnée de l'intersection des Axes X et Y de la courbe.
Xbefore,
YBefore      : Coordonnées graphiques du point précédent affiché.
Pen          : Numéro de stylo ( 1 à 10 max).
Value        : Valeur numérique du point.
Canvas       : Canvas d'affichage (Compatibilité Delphi Borland).
Firspoint    : True si le point est le premier à être affiché sinon False.
LastPoint    : True si le point est le dernier à être affiché sinon False.
```

Graphics.VPU

Cette librairie contient quelques procédures et fonctions graphiques s'appuyant sur l'API Windows.
Elle contient aussi quelques exemples de procédures de **CallBack**.

Pour appeler une procédure de traçage personnalisée (CallBack) utilisez la procédure **TRegisterProc** et passez en paramètre l'adresse de votre procédure de traçage.

Exemple :

```
Var
  PT : Integer;
  PT:=Addr(PtToCircle);  // Récupère le pointeur sur l'adresse de PtToCircle
  TRegisterProc(PT^);   //  Affecte le callback à la procédure PtToCircle
```

Addr(PtToCircle) ne renvoie pas directement l'adresse de la procédure mais une adresse mémoire contenant l'adresse finale de la procédure (indirection ou pointeur).
PT^ signifie le contenu de la mémoire d'adresse contenue dans PT.
Exemple de procédure de traçage en Callback (Extrait de 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 // *** Transtypage de PainStruct **

TGetSelection

Action Retourne un Pointeur sur une structure de type TRECT.
Syntaxe TGetSelection : **Integer**;
Retour Adresse de la structure TRECT de sélection.
Description
Cette fonction renvoie l'adresse de la structure contenant les informations de la dernière sélection à la souris par l'opérateur. L'opérateur peut sélectionner un rectangle (en traits interrompus courts) définissant une zone dans le visionneur de courbes.

Le type Trect est défini dans Graphics.pas comme ci-dessous :

```
   TRECT = Record
               Left   : Integer ;
               Top    : Integer ;
               Right  : Integer ;
               Bottom : Integer ;
       End ;
```

Exemple

```
Uses Graphics ;
Var
  PT :Integer ;
  TR : TREct ;
Begin
   PT:=TGetSelection;
   Move(PT^,TR,Sizeof(TR));
End;
```

Navigateur Web

Le navigateur Web est obtenu par transformation d'une fenêtre (composant window) dans un synoptique en une fenêtre permettant de visionner des pages HTML ou PDF.

```
// ** Crée un navigateur Web dans une fenêtre
Function WEBBROWSERCREATE : Integer;

// ** Permet de visualiser un fichier HTML ou PDF.
Procedure WEBNAVIGATE (Browser : Integer; URL : String);

// *** Aller à la page précédente
Procedure WEBGOBACK ( Browser : Integer);

// *** Aller à la page suivante
Procedure WEBGOFORWARD ( Browser : Integer);

// ** Aller à la page de démarrage
Procedure WEBGOHOME ( Browser : Integer);

// ** Afficher la page de recherche sur le Web
Procedure WEBGOSEARCH ( Browser : Integer);

// ** Permet de ré afficher la page  courante
Procedure WEBREFRESH ( Browser : Integer);

// ** Arrêter l'affichage en cours
Procedure WEBSTOP ( Browser : Integer);
```

Exemple de programme

```
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** : un exemple est disponible sur site web arsoft.

Gestion de la date et de l'heure

Visual I/O contient multiples procédures et fonction pour la gestion des dates et des heures.

Généralités

Visual I/0 utilise le format Real (Type extended) pour le calcul de dates et d'heures.

Les Variables **CDate et CTime** sont disponibles en lecture pour le programmeur. Ces variables globales sont rafraîchies cycliquement tous les 500ms. Un format Chaîne de caractères existe aussi, ce sont les variables **SCDate et SCTime**.

```
CDate  : Real
CTime  : Real;
SCDate        : String];        // *** Ex: 12/04/1994
SCTime        : String;         / *** Ex: 15:45:34
```

A partir de ces quatre variables, il est possible d'afficher la date et l'heure et aussi de calculer d'autres dates ou heures.
Le format de ScDate est dépendant venant de sShortDate de la section [intl] du fichier **WIN.INI**.

Le calcul de la date et de l'heure est ainsi fait :

Heure:= ((Hour X 60 + Min)X 60000) + Sec X 1000 + MSec) / MSecsPerDay;

MSecsPerDay = 24 * 60 * 60 * 1000;

La base de calcul est la milliseconde.
Si par exemple je veux connaître la date courante moins 4 Heures :

```
Var
   TH       : Real;
   DateP    : String;

TH:=((4 *60)*60000)/MSecsPerDay;    // * Calcule le nombre de millisecondes*
Precedent:=(CDate+CTime)-TH;        // Soustrait de la date et de l'heure courante
DateP:=DateTimeToStr(Precedent);    // ** Converti en chaîne de caractères **
```

☞ **Note** : : Pour soustraire des jours il faut les convertir en heures.

Procédures et fonctions concernant la date et l'heure

// Converti une variable du type Real en une Chaîne de caractères
Function **DateTimeToStr** (DateTime: Real) : String;

// Renvoi le Jour de la Semaine
Function **DayOfWeek** (Date: Real): Integer;

// Code une date en fonction des variables Année, Mois, Jour
Function **EncodeDate** (Year, Month, Day: Word) : Real;

// Code une heure en fonction des variables Heures, Minutes, Secondes et Msecondes
Function **EncodeTime** (Hour, Min, Sec, MSec: Word) : Real;

// Renvoi la date et l'heure courante
Function **Now** : Real;

// Converti une chaîne en format de date et Heure
Function **StrToDateTime** (S: string) : Real;

// Converti une chaîne en format d'heure seulement
Function **StrToTime** (S: string) : Real;

// Converti une Time en chaîne de caractères
Function **TimeToStr**(Time: Real) : String;

// Converti une Date en chaîne de caractères
Function **DateToStr**(Date: Real) : String;

// Renvoi la Date et l'heure passée à un format spécifique
Function **FormatDateTime** (Format : String; Dt : Real) : String ;

Spécificateur	**Affichage**
c	Affiche la date, en utilisant le format défini par la variable globale ShortDateFormat, puis l'heure, en utilisant le format défini par la variable globale LongTimeFormat. L'heure n'est pas affichée si la partie fractionnaire de DateTime est égale à zéro.
d	Affiche le numéro du jour sans le préfixer d'un zéro (1 à 31).
dd	Affiche le numéro du jour en le préfixant d'un zéro (01 à 31).
ddd	Affiche le jour abrégé (Dim à Sam) en utilisant les chaînes fournies par la variable globale ShortDayNames.
dddd	Affiche le jour complet (Dimanche à Samedi) en utilisant les chaînes fournies par la variable globale LongDayNames.
ddddd	Affiche la date en utilisant le format fourni par la variable globale ShortDateFormat.
dddddd	Affiche la date en utilisant le format fourni par la variable globale LongDateFormat.
m	Affiche le numéro du mois sans le préfixer d'un zéro (1 à 12). Si le spécificateur m suit immédiatement un spécificateur h ou hh, les minutes sont affichées plutôt que le numéro du mois.
mm	Affiche le numéro du mois en le préfixant d'un zéro (01 à 12). Si le spécificateur m suit immédiatement un spécificateur h ou hh, les minutes sont affichées plutôt que le numéro du mois.
mmm	Affiche le mois abrégé (Jan à Déc) en utilisant les chaînes fournies par la variable globale ShortMonthNames.
mmmm	Affiche le mois complet (Janvier à Décembre) en utilisant les chaînes fournies par la variable globale LongMonthNames.
yy	Affiche le numéro de l'année sous la forme de deux chiffres (00 à 99).
yyyy	Affiche l'année sous la forme de quatre chiffres (0000-9999).
h	Affiche l'heure sans la préfixer d'un zéro (0 à 23).
hh	Affiche l'heure en la préfixant d'un zéro (00 à 23).
n	Affiche les minutes sans les préfixer d'un zéro (0 à 59).
nn	Affiche les minutes en les préfixant d'un zéro (00 à 59).
s	Affiche les secondes sans les préfixer d'un zéro (0 à 59).
ss	Affiche les secondes en les préfixant d'un zéro (00 à 59).
t	Affiche l'heure en utilisant le format fourni par la variable globale ShortTimeFormat.
tt	Affiche l'heure en utilisant le format fourni par la variable globale LongTimeFormat.
am/pm	Utilise le format sur 12 heures avec le spécificateur h ou hh qui le précède. Affiche 'am' pour les heures avant midi et 'pm' pour les heures après midi. Le spécificateur am/pm peut utiliser indifféremment les majuscules ou les minuscules.
a/p	Utilise le format sur 12 heures avec le spécificateur h ou hh qui le précède. Affiche 'a' pour les heures avant midi et 'p' pour les heures après midi. Le spécificateur a/p peut utiliser indifféremment les majuscules ou les minuscules.
ampm	Utilise le format sur 12 heures avec le spécificateur h ou hh qui le précède. Affiche le contenu de la variable globale TimeAMString pour les heures avant midi et le contenu de la variable globale TimePMString pour les heures après midi.
/	Affiche le caractère de séparation de date fourni par la variable globale DateSeparator.
:	Affiche le séparateur des heures fourni par la variable globale TimeSeparator.
'xx'/"xx"	Les caractères compris entre des simples ou doubles guillemets sont affichés tels quels et n'affectent pas le format.

Les spécificateurs de format peuvent être indifféremment en majuscules ou en minuscules ; le résultat obtenu est identique.
Si la chaîne fournie par le paramètre Format est vide, la date et l'heure sont implicitement formatées avec le spécificateur de format 'c'.
Exemple :
L'exemple suivant affecte la chaîne 'Rendez-vous Vendredi, Février 13, 1997 à 10:30 AM' à la variable S.

```
S := FormatDateTime('"Rendez-vous " dddd, mmmm d, yyyy, ' +
  '"à" hh:mm AM/PM', StrToDateTime('13/2/97 10:30am'));
```

Gestion de fichiers Textes

Un fichier texte ou (le plus souvent abusivement) fichier ASCII, est un fichier dont le contenu représente uniquement une suite de caractères alphanumériques. Il faut noter que l'espace et le retour à la ligne sont considérés comme des caractères au même titre qu'une lettre, un chiffre ou un signe de ponctuation. Ce type de fichier est lisible avec notepad ou wordpad par exemple.
Un fichier « non texte » est appelé « fichier binaire », dans le sens où les bits contenus dans le fichier ne peuvent pas être représentés pas une simple suite de caractères.
La dénomination fichier ASCII est souvent abusivement utilisée à propos de fichiers texte n'utilisant pas le codage de caractères ASCII. Ces fichiers sont accessibles ligne par lignes. Ces lignes peuvent être de différentes longueurs.
Par exemple pour accéder à la ligne N°3 dans un fichier texte il faut d'abord lire la ligne 1 puis la ligne 2 avant d'accéder à la ligne N°3. Le fichier texte est dit à accès séquentiel.

Liste des instructions

// Concatène 2 fichiers}
Function **FileConcat** (Source, Destination : String) : Boolean;

// Copie un fichier vers un autre fichier}
Function **FileCopy** (Source, Destination : String) : Boolean;

// Efface un fichier}
Function **FileDelete** (FileName: String) : Boolean; *// ** Depuis Version 7 est dans Gestfile.VPU*

// Test si un fichier existe}
Function **FileExists** (FileName : String) : Boolean;

// Renomme un fichier}
Function **FileRename** (OldName, NewName: String): Boolean;

// Ouvre en ajout un Fichier Texte}
Procedure **FileTextAppend** (Var F: Integer; Name : String);

// Ferme un Fichier Texte}
Procedure **FileTextClose** (Var F: Integer);

// Ouvre en Mode Lecture un Fichier Texte}
Procedure **FileTextOpen** (Var F: Integer; Name : String);

// Lit la ligne suivante dans le fichier text ouvert}
Procedure **ReadLn** (Var F : Integer; St : String);

// Ecrit une chaîne à la fin du fichier text ouvert
Procedure **WriteLn** (Var F : Integer; St : String);

// Test si la fin de fichier est atteinte
Function **EOF** (Var F : Integer) : Boolean ;

Exemple de lecture d'un fichier Texte

```
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;
```

Accès aux variables globales par fonctions

Function **GetVpuValue**(NomVar : String) : String;
*// ** Renvoie la valeur de la variable demandée sous forme de chaîne de caractères.*
Procedure **SetVpuValue**(NomVar, Value : String) ;
*// ** Fixe la nouvelle valeur de la variable de nom contenu dans NomVar.*

```
Exemple :               SetVpuValue( 'W0','123') ;
                        SetVpuValue( 'B0,'TRUE') ;
```

Function **GetVpuAddr**(NomVar : String) : Integer;
*// ** Renvoie l'adresse mémoire de la variable spécifiée.*
Les variables sont rangées en mémoire consécutivement (Packed) en récupérant l'adresse du variable et connaissant *les déclarations des autres variables il est alors possible de manipuler toutes les variables par block et non une par une.*
Function **GetProjectDir** : String; *// ** Renvoi le **chemin** de l'application courante.*

Autres
Procedure **VPLOAD** (VPUFileName : ShortString);
*// *** Charge une librairie ou programme compile par Visual I/O ou Visual PLC ****
Exemple :
VPLOAD ('TACHE1.VPU'); *// *** Extension .VPU nécessaire*

Function **VPUNLOAD** (VPUFileName : ShortString) : Integer;
*// *** Décharge de la mémoire une librairie ou un programme compile ***
VPUNLOAD ('TACHE1'); *// *** Sans extension*

Function **ExtractRes** (**FileNameinEXE :** String; **DestinationFile** : String) : **Boolean**;
*// ** Permet d'extraire un fichier de l'exécutable créée avec Visual I/O.*

FileNameinEXE Nom du fichier stocké dans l'exécutable.
DestinationFile Nom et chemin sur le disque du fichier après extraction.
Retour Extraction correctement effectuée.

Exemple :
```
Ok:=ExtractRes ('Menu.HTM','C:\Doc\Menu.htm') ;
```
'Menu.HTM' est le nom de la ressource dans l'exécutable courant.
L'exécutable mentionné est le programme contenant cette instruction et non pas un exécutable externe.

☞ **Note** : Afin de visualiser les ressources dans un exécutable fait par Visual I/O, utilisez l'utilitaire SCANRESEXE.EXE

Procedure **WinDialog** (Value : Boolean);
*// *** Passe le synoptique courant en mode Dialog, c'est-à-dire que les programmes du synoptique ne seront exécutés que si le synoptique à le focus (barre de titre active)*

Envoyer un Email par programme

Function **XSendEmail** (MailFrom, MailRecipient, MailSubject, MailBody, SmtpHost : **String**) : **String**;

MailFrom : Provenance de l'email
MailRecipient : Email de destination.
MailSubject : Sujet de l'email
MailBody : Texte de l'email
SmtpHost : Serveur Smtp utilise pour envoyé l'email. (Courrier sortant voir messagerie)
Exemple *Lancer le client de messagerie **Outlook Expres**s.*
*Cliquer sur **Outils-Comptes**.*
*Choisir **Courrier***
*Cliquer sur votre **compte**, puis cliquer sur **Propriétés et onglet serveurs***
Recopier le champ Courrier sortant (SMTP).
Retour : Chaine vide si ok sinon erreur et retour du texte de l'erreur

Exemple :
XSendEmail('Application Viaual I/O', 'contact@hotmail.com', 'Felicitations', 'Envoyer un email est très facile', 'smtp.hotmail.com');

Les menus pop up dynamiques

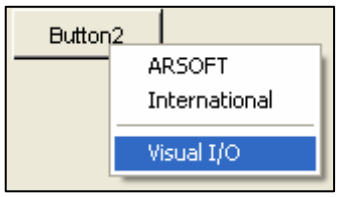
Ces procédures et fonctions permettent de créer dynamiquement des menus Popup (surgissant par le clic droit de la souris). Cette fonctionnalité est intéressante lorsqu'il est nécessaire de changer de commande en fonction du contexte ou de la figure sur laquelle la souris se trouve. Pour le test utilisez le même composant et instructions qu'un menu principal.

Function **POPUPMenuCreate** : Integer;
*// ** Crée un menu pop vide*

Procedure **POPUPMenuDestroy** (Handle : Integer);
*// ** Détruit un menu popup*

Procedure **POPUPMenuAddChild** (Handle : Integer; Text : String)' *// **** Text = Texte + , +Identificateur;*
*// ** Rajout des items au menu popup désigné*

```
// *** 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;
```



Charger et sauver un contexte d'ActiveX

*// ** Permet de connaître le nom de l'activeX derrière lequel cette function est exécutée.*
Fonction **OCXGetClassName** : String;

*// *** Sauve les propriétés courantes de l'activeX afin de pouvoir les recharger éventuellement*
*// *** Propriétés graphiques du moment (couleurs, fontes, tailles etc..*
Procedure **SavePropOcx** (Nom : String);

*// *** Charge dans les propriétés de l'activeX, les propriétés sauvegardées auparavant*
Procedure **LoadPropOcx** (Nom : String);

Changer la fonte d'un texte et son style

*// ** Fixe le nom de la fonte du contrôle graphique*
Procedure **SetFontName** (FName : String);

*// *** Change le style de la fonte du contrôle graphique*
Procedure **SetFontStyle** (FStyle: Integer);

Fstyles : Bold = 0
Italic = 1
Underline = 2
StrikeOut = 3

Gestion des Bases de données avec BDE

Visual I/O intègre le BDE (Borland DataBase Engine) et permet de naviguer, trier, ajouter, effacer... des données ou des enregistrements dans des tables de type PARADOX, DBASE ou autres selon le type de driver installé et choisi.
La création et la modification de tables peut se faire à l'aide de l'utilitaire DataBase-DeskTop (ou par requête Sql).

Pour une manipulation très rapide de **Tables DBAse 3+** uniquement vous pouvez utiliser la librairie **Microbase.VPU**.
Cette librairie ne permet que la navigation, insertion, effacement, ajouts et tris simples sur une table Dbase 3+. Elle a cependant l'avantage d'être extrêmement rapide car elle gère la table Dbase 3+ comme un fichier à accès direct. Dans beaucoup de cas Microbase suffit. DBASE 3+ est compatible avec tous les logiciels de bureautique.

Les tables et les structures de données

La création de table permet de définir la structure d'une table, c'est à dire l'organisation et le type des données. Pour créer modifier ou visualiser des tables, utilisez le DataBase-DeskTop.

Les Index ou Clés

Le système d'indexation est l'épine dorsale d'un SGBD (Système de Gestion de Base de Données). Sans les possibilités qu'il offre pour effectuer des localisations rapides, fournir des ordres de tri et opérer des liaisons de tables, les données contenues dans celles-ci ne seraient qu'une masse d'octets inerte et inexploitable.

Les mots **Clé** et **Index** sont synonymes:
La clé primaire d'une table (PARADOX) détermine l'ordre séquentiel des données ainsi que la possibilité d'établir des liens avec d'autres tables.
Une clé primaire simple se définit sur le premier champ de la table.
Une clé primaire composite se définit sur les N premiers champs de la table.

Qu'il s'agisse d'une clé simple ou composite, cette clé doit être unique, c'est à dire que l'information apparaissant dans chaque clé d'enregistrement ne peut être dupliquée. Cela signifie que :
Pour une clé simple, le même champ de deux enregistrements différents ne peut pas avoir la même valeur. Par exemple, une table possédant une clé simple sur le champ «Nom» et un champ «Prénom» ne faisant pas partie de cette clé ne pourra accepter les individus Jean Dubois et Jacques Dubois, parce que «Dubois» ne peut être la clé que d'un seul enregistrement.
Pour une clé composite, l'ensemble des valeurs contenus dans la clé doit être globalement unique.
En reprenant le même exemple que précédemment, mais avec une clé primaire sur «Nom» et «Prénom», la table pourra contenir à la fois Jean Dubois et Jacques Dubois, ou bien Jean Dubois et Jean Durant, mais pas deux fois Jacques Dubois.

Programmation

Une **base de donnée** est en général un **répertoire** contenant des tables à des formats tel que PARADOX, DBASE, Acces etc..
Visual I/O permet de naviguer, modifier, insérer etc... des enregistrements dans ces tables. Cette navigation est possible à travers un objet créé par programme. Cet objet met à votre disposition des méthodes (Fonctions et Procédures) .
L'objet avant tout, doit être créé et renseigné sur la base de données qu'il doit servir et sur la table sur laquelle il doit travailler. Les fonctions de cet objet sont contenues dans la VPU BDE.VPU.

Exemple de déclaration d'un objet d'interface de Base de données :

Dans les variables commune de votre synoptique déclarez :

```
Uses BDE ; { Les fonctions sont dans cette VPU *
Var
  MaBase : Integer;  {*** Mabase est un Handle de Table premiers
```

Dans un programme quelconque :

```
If Start_Visu then
   Mabase:= DBCreate('C:\Base\Cumul.DBF'); // Base de donnée dans répertoire
                                           // C:\Base

If End_Visu Then {*** En quittant le programme je ferme la base de données
      DBClose(MABase);
```

Note :
Tant que la base de données n'est pas fermée (DBClose) les nouvelles données introduites ne sont pas véritablement écrites sur le disque.
Plusieurs canaux peuvent être ouvert en même temps.
Fermer alors des canaux (DBclose) pour retrouver un nombre normal de fichiers ouverts.

Liste des Instructions

Function **DBOk** (Base : Integer) : Boolean; *// Test si la base et la table en cours sont correctement ouvertes*
Procedure **DBClose** (Base : Integer); *// Ferme la base et la table en cours*
Procedure **DBCreate** (Table,User,Password: String); *// Crée une interface avec une base de données et une table User et password sont des paramètres facultatifs*
Function **DBAppend** (Base : Integer): Boolean; *// Ajoute un nouvel enregistrement à la fin de la table en cours*
Function **DBBof** (Base : Integer): Boolean; *// Indique si la table est sur le premier enregistrement*

// Valide ou non le message de confirmation d'effacement d'un enregistrement.
Function **DBConfirmDel** (Base : Integer,C : Boolean);
Function **DBDelete** (Base : Integer): Boolean; *// Efface l'enregistrement en cours*
Function **DBDeleteTable** (Base : Integer): Boolean; *// Détruit complètement la table en cours*
Function **DBEmptyTable** (Base : Integer): Boolean; *// Efface tous les enregistrements de la table courante*
Function **DBEof** (Base : Integer): Boolean; *// Indique si la table est sur le dernier enregistrement de la table*

// Exécute une requête SQL ou QBE stockée dans un fichier
Function **DbexecQuery** (Base : Integer ; Name : String) : Boolean;
// Recherche le premier champ contenant l'élément demandé
Function **DBFind** (Base : Integer ; S: String):Boolean;
// Recherche le premier champ se rapprochant de l'élément demandé
Function **DBFindNearest** (Base : Integer ;S : String) : Boolean;

Function **DBFirst**(Base : Integer) : Boolean; *// Positionne la table en cours sur le premier enregistrement*
Function **DBGetCount** (Base : Integer) : LongInt; *// Renvoie le nombre d'enregistrements dans la table*

// Retourne la position de l'enregistrement courant dans la table en cours
Function **DBGetCurrentPosition** (Base : Integer): LongInt;

// Renvoie sous forme de chaîne de caractères le contenu du Nième
Function **DBGetField** (Base : Integer ; N:Integer) : String;
// Renvoi le contenu d'un champ de la table en cours
Function **DBGetFieldByName** (Base : Integer ; N:String) : String;

// Renvoi la dernière erreur rencontrée dans le BDE
Function **DBGetLastError**: String; *// Commun à toutes les Bases ouvertes*
// Renvoi le nom d'un index de la table en cours
Function **DBGetNameField** (Base : Integer ; N:Integer) : String;
// Renvoi le nombre de champs de la table en cours
Function **DBGetNameIndex** (Base : Integer ; N:Integer) : String;
Function **DBGetNFields** (Base : Integer) : Integer; *// Renvoi le nombre indexes de la table en cours*
Function **DBGetNIndexs** (Base : Integer) : Integer; *// Renvoi le nombre indexes de la table en cours*
Function **DBGotoBookMark** (Base : Integer): Boolean; *// Renvoi au dernier Marque page*
Function **DBInsert** (Base : Integer): Boolean; *// Insert un enregistrement à la position courante de la table*
// Renvoi les Indexs en cours de la table courante
Function **DBGetindexFieldName** (Base : Integer): String ;
// Fixe les Indexs valides de la table courante
Function **DBSetindexFieldName** (Base : Integer ; IndexsNames : String);

Function **DBGETindexName** (Base : Integer): String; // *Renvoi l'index de tri en cours sur la table courante*
// Renvoi l'index de tri en cours sur la table courante
Procedure **DBSETindexName** (Base : Integer ; IndexName: String);
Function **DBLast** (Base : Integer) : Boolean; *// Positionne la table en cours sur le dernier enregistrement*
Function **DBModify** (Base : Integer): Boolean; *// Modifie l'enregistrement courant de la table en cours*
// Avance ou recule de N Enregistrements dans une table
Function **DBMoveBy** (Base : Integer ; N : Integer) : Boolean;
Function **DBNext** (Base : Integer) : Boolean; *// Avance à l'enregistrement suivant d'une table*
Function **DBPackTable** (Base : Integer) : Boolean; *// Optimise l'espace du table DBASE ou PARADOX.*
Function **DBPrior** (Base : Integer): Boolean; *// Recule d'un Enregistrement dans une table*
// Ecrit un champ particulier dans le buffer mémoire de l'enregistrement courant dans une table
Function **DBPutField** (Base : Integer ; N : Word; S: String) : Boolean;
// Ecrit un champ particulier en le désignant par son nom, dans le buffer mémoire de l'enregistrement courant dans une table
Function **DBPutFieldByName** (Base : Integer ; Name : String; S: String) : Boolean;

Procedure **DBQAdd** (Base : Integer ; S : String); *// Ajoute une ligne dans une requête Sql*
// Indique si la table de réponse du requête est sur le premier enregistrement
Function **DBQBof** (Base : Integer): Boolean;
// Efface l'enregistrement en cours dans une table de réponse SQL
Function **DBQDelete** (Base : Integer): Boolean;
// indique si la table réponse d'une requête est sur le dernier enregistrement
Function **DBQEof** (Base : Integer): Boolean;
{Exécute un remplacement d'une variable dans un fichier requête
Function **DBQExchange** (Base : Integer ; Name,S,R : String) : String;
// Positionne la table réponse en cours sur le premier enregistrement
Function **DBQFirst** (Base : Integer): Boolean;
// Renvoi le nombre d'enregistrements dans la table réponse
Function **DBQGetCount** (Base : Integer): LongInt;

// Renvoi sous forme de chaîne de caractère le contenu du Nième champ de l'enregistrement courant dans la table de réponse
Function **DBQGetField** (Base : Integer ; N:Integer) : String;

// Renvoi le contenu d'un champ de la table réponse en cours
Function **DBQGetFieldByName** (Base : Integer ;N:String) : String;

// Renvoi le nom d'un champ de la table réponse en cours
Function **DBQGetNameField** (Base : Integer ; N:Integer) : String;
// Renvoi le nombre de champs de la table réponse en cours
Function **DBQGetNFields** (Base : Integer) **:** Integer;
// Positionne la table réponse en cours sur le dernier enregistrement
Function **DBQLast** (Base : Integer) : Boolean;
// Modifie l'enregistrement courant de la table réponse en cours
Function **DBQModify** (Base : Integer): Boolean;

// Avance ou recule de N Enregistrements dans une table réponse
Function **DBQMoveBy** (Base : Integer ;N : Integer) : Boolean;
Function **DBQNext** (Base : Integer): Boolean; *// Avance d'un Enregistrement dans une table réponse*
Function **DBQPrior** (Base : Integer): Boolean; *// Recule d'un Enregistrement dans une table réponse*

// Ecrit un champ particulier dans le buffer mémoire de l'enregistrement courant d'une table réponse
Function **DBQPutField** (Base : Integer ;N : Word; S: String) : Boolean;
Procedure **DBResetRange**(Base : Integer); *// Remet à zéro une zone de travail dans la table en cours*
Function **DBSetBookMark** (Base : Integer): Boolean; // Marque un endroit de la table en cours
// Fixe un nouveau message de confirmation d'effacement
Procedure **DBSetMessageDel** (Base : Integer ;S : String);
// Fixe la nouvelle clé (index) de recherche dans la table courante
Procedure **DBSetNindex** (Base : Integer ;N : Byte);
// Positionne le curseur (Pointeur) sur un enregistrement particulier
Function **DBSetPosition** (Base : Integer ;P : LongInt) : Boolean;

// Fixe une zone de travail dans la table courante
Function **DBSetRange** (Base : Integer ; St,Se : String; Exclusive : Boolean) : Boolean;
Function **DBSortTable** (Base : Integer) : Boolean; *// Trie une table PARADOX ou DBase non indexée*

// Interdit ou autorise l'affichage des messages d'erreurs concernant les bases de données
Variable **StopDBMsg** : Boolean; *// Variable commune à toutes les Bases ouvertes*

Requêtes QBE & SQL

Visual I/O peut traiter des requêtes SQL & QBE sur vos tables. Vous pouvez en registrer les instructions Qbe ou Sql dans un fichier. Visual I/O traite en général aussi bien vos requêtes Qbe ou Sql.

Sauvegarde dans un Fichier .QBE ou .SQL

Après avoir défini votre Qbe (et éventuellement testé celui-ci), vous devez le sauvegarder dans un fichier d'extension QBE ou SQL pour qu'il soit exploitable par Visual I/O grâce à la commande **DBexecQuery** .

Exemple de requête SQL sauvegardée :

```
SELECT DISTINCT "C:\BDE\EXEMPLES\TABLES\STOCK.DB"."Stock No", Model
FROM "C:\BDE\EXEMPLES\TABLES\STOCK.DB"
WHERE
("C:\BDE\EXEMPLES\TABLES\STOCK.DB"."Stock No" > 10)
ORDER BY "C:\BDE\EXEMPLES\TABLES\STOCK.DB"."Stock No", Model
```

Exemple de requête QBE sauvegardée :

```
C:\BDE\EXEMPLES\TABLES\STOCK.DB     | Stock No     | Model  |
                                    | Check >10    | Check  |
```

Requêtes QBE & SQL Dynamiques

Après avoir formaté et sauvegardé vos requêtes, vous pouvez définir des variables dans cette requête qui seront affectées lors du lancement de celle-ci.
Dans l'exemple précédent portant sur une requête QBE nous avons un test figé sur le champ Stock No. Ce test est de rechercher tous les Stock No supérieur à 10.

Vue :

```
C:\BDE\EXEMPLES\TABLES\STOCK.DB     | Stock No     | Model  |
                                    | Check >10    | Check  |
```

Il serait intéressant d'affecter dynamiquement la valeur de comparaison (ici 10) par une autre valeur.
Pour ce faire nous allons remplacer la valeur 10 par une variable de nom %%.

Modification et sauvegarde du QBE sous éditeur de texte:

```
C:\BDE\EXEMPLES\TABLES\STOCK.DB     | Stock No     | Model  |
                                    | Check >%%    | Check  |
```

Dans votre programme vous devez affecter la variable %% par une valeur réelle avant d'envoyer la requête Qbe (Même principe en SQL). Pour Modifier cette variable vous devez utilisez la commande **DbQExchange.** Imaginons que vous voulez affecter la variable %% par la valeur 53. La commande sera du type :

```
NOUVEAU_FICHIER:=DbQExchange(Mabase,'REQUETE.QBE','%%','53');
```

Cette ligne de programme échange dans le fichier REQUETE.QBE la variable nommée %% par la valeur 53 et renvoi le nom d'un fichier contenant cette requête modifiée.

Pour indication et éventuellement vérification, le nouveau fichier sera stocké dans le répertoire courant sous le nom **A.QBE** ou **B.QBE**.

```
C:\BDE\EXEMPLES\TABLES\STOCK.DB     | Stock No     | Model  |
                                    | Check >53    | Check  |
```

Il est possible alors d'appeler la commande d'exécution de la requête :

DBexecQuery(Mabase, NOUVEAU_FICHIER);

Vous pouvez avoir à rectifier plusieurs variables dans une requête.
Exemple :

C:\TABLES\STOCK.DB | Stock No | Model | Part No |
| Check >%% | Check dv.| Check >### |
Ici nous avons déclaré 2 Variables %% et ###, il faut donc les remplacer par programme avant de lancer la requête.

%% sera remplacé par la Valeur 53 et
sera remplacé par la Valeur 120.

Programme :

NOUVEAU_FICHIER:= **DbQExchange**(Mabase,'REQUETE.QBE','%%','53');
NOUVEAU_FICHIER:= **DbQExchange**(Mabase,NOUVEAU_FICHIER,'###','120');
DBexecQuery(Mabase,NOUVEAU_FICHIER);

Première ligne = échange de toutes les variables %% contenues dans le fichier REQUETE.QBE par la valeur 53 et renvoi du nom du fichier contenant la modification (Requête.Qbe ne change pas).

Deuxième ligne = échange de toutes les variables ### contenues dans le fichier de nom stocké dans la variable NOUVEAU_FICHIER par la valeur 120 et renvoi du nom du fichier contenant la modification dans la variable NOUVEAU_FICHIER.

Troisième ligne = Exécution de la requête décrite dans le fichier de nom stocké dans la variable NOUVEAU_FICHIER.

Note : en aucun cas la requête source (ici REQUETE.QBE) n'est modifiée. Les modifications sont portées dans les fichiers créés par la commande **DbQExchange.**

Nom des Variables Dynamiques :

Il est possible d'imaginer n'importe quel nom de variable exemple :
$$ ****))) 777 AZE VISUALIOWINDOWS
Cependant la procédure **DbQExchange** pourra confondre ## et ###, ou VISUAL et VISUAL_IO.
Il faut donc éviter de choisir les mêmes caractères pour définir une variable dynamique.

Réponses aux requêtes QBE & SQL

Après avoir exécuter la commande **DBexecQuery** Visual I/O crée une table réponse virtuelle contenant le résultat de la requête effectuée. Vous avez accès alors à un jeu d'instructions très proche du jeu d'instruction traditionnel pour interroger la table de réponse.

Le jeu d'instruction est toutes les fonctions ou procédure commençant par :

DBQ....

Constitution de Requêtes Sql par programme :

Il est possible de créer une requête Sql dynamique par programme en utilisant la procédure **DBQAdd** ,
Chaque ligne sera ajoutée à une liste interne à exécuter.

Exemple :

```
{*** Dans Variables Locales **
MaBase: DataBase;
{***** Dans programme d'une figure ****
If Start_Visu Then
    MaBase:=DataBase.Create(''); //***
```

```
If Button Then
Begin
   DbQadd(Mabase,'SELECT Libelle');
   DbQadd(Mabase, 'FROM "ESSAI.DB" );
   DbQadd(Mabase, 'WHERE');
   DbQadd(Mabase, 'LIBELLE= "Visual IO "');
   if DbExecQuery( Mabase, '') Then  {*** Exécution de la requête constituée *
    {**** Traitement ******
End;
```

Exemples

Récupération de la valeur d'un champ d'une table :

```
{****** Dans Variables Locales *********
MaBase: DataBase;
{***** Dans programme d'une figure Texte ********
If Start_Visu Then
Begin
  MaBase:=DataBase.Create('C:\Base\Clients.dbf'); {* répertoire contenant la
table Si un mot de passe et un utilisateur sont nécessaire ajoutez 2 paramètres
End;
If Button   Then SetText(Mabase.DbGetField(Mabase,1));
If End_Visu Then DBClose(Mabase);
```

Affectation de champs d'une table :

```
{*** Dans Variables Locales **
MaBase: DataBase;
{***** Dans programme d'une figure ****
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 procédure DBCreate

DBCreate Procédure

Action Crée une interface avec une base de données et une table.

Syntaxe <*Labase*>:= DBCreate(**Path** : String; User, Password : **String**);

Description
Cette fonction permet d'initialiser un Handle de type DataBase et de le renseigner sur la base de données et la table que l'on veut adresser. On peut initialiser autant d'objets Database que l'on veut mais on ne peut l'utiliser qu'une seul fois tant que l'on n'a pas appliqué sur cet objet la fonction DBClose. A chaque **DBCreate** utilisé, correspond un **DBClose**.
L'instruction DBCreate peut être utilisé au lancement de l'application et l'instruction DBClose utilisée à la disparition de l'application. Cependant, on utilise couramment DBCreate et DBClose à chaque opération sur une table afin de garantir la sauvegarde correcte des données. Dbcreate peut aussi recevoir (option) 2 autres paramètres correspondant à l'utilisateur (user) et son mot de passe(Password). Si c'est 2 paramètres ne sont pas renseignés le compilateur place 2 chaines vides à la place.

Pour disposer d'un accès sur une table d'une base de données, il faut :

1. Déclarer un objet de type DataBase dans les variables Locales.
Exemple: MaBase : **Integer**; *{Handle MaBase du type DataBase.}*
2. Initialiser cet Objet.
Exemple :
```
If Start_Visu Then MaBase:= DBCreate('C:\Examples\Tables');
```
3. Refermer l'interface de la base de donnée et restituer la mémoire
Exemple :
```
If End_Visu Then DBClose;
```

LimitesA un DBCreate correspond un seul DBClose.
Windows limite le nombre de fichiers ouvert en même temps.

Extensions Alias:
L'instruction DBCreate permet de travailler avec les Alias définis en configuration (Icône Configuration Database Engine vu en début de ce chapitre).
La procédure DBCreate permet d'utiliser un alias déjà défini.
Exemple :

```
Mabase:= DBCreate('TestExcel\Donnes');      //* travaille avec l'alias TestExcel
                                            //* Travaille avec la table Donnes.Xls
```

Dans ce cas la base de données adressée est le chemin contenu dans l'alias TestExcel et la table est DONNEES sans type de table correspondant puisque l'Alias connaît déjà ce type de table.

L'Alias est intéressant dans 2 cas :

Il permet de changer de base de données (Répertoire) sans retoucher au code du programme puisque l'alias donne le répertoire cible. Il suffit simplement de reconfigurer le DataBase Engine en lui donnant le nouveau répertoire cible. Cette fonctionnalité est intéressante lorsque vous travaillez en réseau. Vous pouvez tester en local sur votre PC votre programme puis en final affecter le chemin réseau final sans avoir à retoucher votre code programme.

Il permet de travailler avec des drivers ODBC. Une base de donnée dans Visual IO ne peut travailler sur une table autre que DBASE et PARADOX qu'à travers ODBC et un ALIAS correspondant. L'exemple si dessus permet d'attaquer une table EXCEL à travers un Alias qui lui travaille avec un driver ODBC.

Chargement déchargement manuel d'une DLL

La DLL BDE est l'illustration du chargement et déchargement manuel d'une DLL. Ce mécanisme ne devrait pas être utilisé car Visual I/O utilise déjà ce mécanisme lorsque le programmeur à utilisé la syntaxe pour appel de procédures et fonctions dans une DLL (Voir chapitre suivant).
Pour le chargement d'une Librairie utilisez la commande LoadLibrary de l'api Windows.
Pour retrouver l'adresse du procédure ou d'une fonction dans cette DLL utilisez la commande GetProcAddress.
Pour décharger de la mémoire la librairie utilisez la commande FreeLibrary.

Exemple : (Tiré de BDE.PAS).

```
//** chargement mémoire de la DLL
 HDle :=LoadLibrary(PChar('Bdext'));
//** Récupère l'adresse de la procedure DBGETIndexName dans la DLL
 AD:=GetProcAddress(Hdle,PChar('DBGetIndexName'));
//* Remplace l'adresse de la procédure Visual I/O
 Move(Ad,DBGetIndexName,4);
//* Décharge de la mémoire la DLL ses procédures sont détruites
 Freelibrary(HDLE);
```

Appel de procédures et de fonctions d'une DLL

Visual I/O permet d'appeler des procédures et des fonctions qui sont contenues dans des librairies dynamiques appelées communément **DLL**.
Cette interface est intéressante dans le cas ou vous recevrez des librairies accompagnant des cartes scientifiques par exemple, ou que vous décidiez d'enrichir Visual I/O par des fonctionnalités écrites sous un autre langage (par exemple en Visual Pascal).

Pour utiliser ces procédures et fonctions contenues dans une DLL il faut déclarer la structure de la fonction ou la procédure dans la partie INTERFACE d'une librairie. La librairie servant alors de lien commun entre Visual I/O et la DLL.

Examinons la déclaration suivante :

```
Unit
Interface
   Function  MessageBeep(Utype : Integer) : Integer; StdCall 'user32.dll';
Implementation

Begin
End.
```

Cette librairie VPU permet de mettre à disposition des autres VPU de Visual I/O une nouvelle fonction nommée MessageBeep.
Cette fonction n'est pas implémentée dans la VPU elle-même mais dans une DLL nommée User32.dll.
Il faut lui passer un paramètre nommé Utype de type Integer.
StdCall permet de signifier au compilateur qu'il doit respecter le mode standard de passage de paramètres défini par Microsoft pour les applications en 32Bits.

Dans le cas de MessageBeep, la fonction de User32.dll est aussi nommée MessageBeep.
Attention respectez scrupuleusement les majuscules/Minuscules dans les déclarations.

La déclaration suivante ne fonctionnerai pas :

```
Function  MESSAGEBEEP(Utype : Integer) : Integer ; StdCall 'user32.dll';
```

Redéfinir le nom de la fonction contenue dans une DLL

Vous pouvez redéfinir le nom d'interface avec la DLL. La déclaration suivante le permet :

```
Unit
Interface
  Function  Bip (Utype : Integer) : Integer; StdCall 'user32.dll' name 'MessageBeep';
Implementation
```

Vous définissez une nouvelle fonction exploitable dans Visual I/O qui est Bip et qui correspond à MessageBeep dans user32.dll.

Localisation d'une DLL

Il est possible en l'absence de chemin que l'exécutable créé par Visual I/O ne trouve pas la DLL nommée.
Copiez vos DLL dans un répertoire toujours connu de Windows.
Vous pouvez par exemple copier vos DLL dans WINNT (Version NT) ou Windows (version 98,2000 XP).
L'exécutable de votre application localisera à coup sur cette DLL.

Vous pouvez aussi indiquer un chemin complet indiquant la localité de la DLL.

```
Function  Go(Utype : Integer) : Integer ; StdCall 'C:\Proj1\Essai.DLL' name 'Demarre';
```

Visual I/O met à disposition de nombreux types de données pour que l'interface avec des DLL sont le plus compatible possible. Cependant le problème le plus souvent rencontré est celui des chaînes de caractères. En langage C par exemple le type souvent utilisé est PChar (Pointer sur chaine de caractères finie par le caractère Nul 0). En pascal la chaine de caractères est différente dans le fait que le premier caractère est la longueur. Afin de transtyper (changer de type pour que se soit compatible) utilisez la fonction PChar de la librairie Windws.pas (dont le source est fourni).

Exemple de passage de texte à une DLL windows.

Exemple : ExternFunc (PChar ('Bonjour')); PChar ('Bonjour') renvoi l'adresse du premier caractère de la chaîne de caractères Bonjour. La fonction PChar ajoute aussi (si ce n'est pas le cas) un caractère nul (0) à la fin de la chaîne.

Pointeurs et transtypages

Pointeurs

Un pointeur est une variable dont le le contenu n'est pas une donnée mais une adresse mémoire (emplacement dans la mémoire).
L'utilité d'un pointeur est d'accéder à une variable par son pointeur, c'est à dire son adresse mémoire.
Le type pointer n'existe pas en visual I/O car il est équivalent à Integer (32 bits). Cependant par soucis de lisibilité, le type pointer à été défini dans Sysplc.
Type Pointer = Integer;

Exemple :

```
Var
  I     : Integer;
  PInt  : Integer;
Begin
  PInt:=Addr(I);
```

PInt contient l'adresse mémoire de la variable I.
Par convention, lorsque l'on créé généralement un pointeur, on commence le nom du pointeur par la lettre "P" pour distinguer plus facilement une variable et un pointeur.

Pour accèder à une variable à partir de son pointeur, il suffit dans un premier temps de récupérer l'adresse de la variable grâce à la fonction "Addr" (ou l'opérateur "@") que l'on a vu précédemment.
Puis dans un deuxième temps, de déréférencer le pointeur, c'est à dire de prendre en compte le contenu (l'adresse de la variable "i") que désigne le pointeur. Pour déréférencer le pointeur, il suffit de spécifier le nom du pointeur suivit de
l'opérateur "^". On accède ainsi à la valeur de l'adresse correspondante.
Exemple :

```
Begin
  PINT^:=10;
  PINT^:= PINT^+ 1;
Equivaux à
Begin
   I:=10;
   I:=I+1;
```

Allocation mémoire - GetMem et FreeMem

Ces 2 instructions permettent d'allouer de la mémoire et de restituer la mémoire allouée.
Vous pouvez alors allouer (GetMem) dynamiquement des variables puis les détruire après utilisation (FreeMem).

```
Var
  PTByte : Pointer;  //** Pointer sur une zone mémoire
  PTdest : Pointer;  //** Autre pointer

Begin
  GetMem   (PTByte,80);    //** Crée un pointer sur une zone mémoire de 80 bytes
  FillChar (PTByte,80,0);  //** Rempli La zone mémoire 80 bytes avec 0
```

Copy d'une zone mémoire dans une autre

```
  GetMem  (PTDest,200); //*** Crée un pointer sur une zone mémoire de 200 bytes
  Move    (PTByte,PTDest,80); // la procédure Move Copie la première zone
                              dans la 2ème
```

Libération de la mémoire

```
  FreeMem  (PTint,80);   //** Libère la première zone mémoire 80 octets
```

GetMem : Range dans PTByte l'adresse de début de la mémoire allouée.
FillChar : Remplit une succession d'octets à par tire de l'adresse contenue dans PTByte avec la valeur spécifiée.
FreeMem : FreeMem libère une zone mémoire dont l'adresse de début est contenu dans PTint et d'une taille donnée

Transtypage

Transtyper permet de changer ou d'affecter un autre type à une variable.
On peut modifier le type d'une variable à l'exécution en la précédant du nom d'un autre type puis de la variable entre parenthèses. Il faut éviter autant que possible cette action qui conduit souvent à des erreurs, mais dans certains cas il n'y a pas d'autres solutions.

Exemple :

```
  PTByte : Pointer  //** Ce type de pointeur, peut pointer sur tout.
```

Pour accéder à un élément du tableau de Byte, il convient de transtyper l'élément pour pouvoir accéder à ses champs.

```
TArrayByte(PTByte)[1]:=10;
//** On dit ici au compilateur que la variable PTByte est du type Array of Byte.
```

Attention : //** Tous les transtypages nommés Tarrayxxx commencent à l'élément 1

TArrayByte est défini comme ceci dans 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;
```

Si nous voulions utiliser la zone mémoire PtByte comme une zone de Reals, il faudrait coder comme ceci

```
TArrayReal(PTByte)[1]:=345.67;
TArrayReal(PTByte)[2]:=PI;
```

La zone mémoire peut être manipulée à souhait. Exemple

```
Var
  B1 : Byte;
  R1 : Real;
Begin
  TArrayByte(PTByte)[1]:=10;   //*** Comme une zone de bytes
  B1:=TArrayByte(PTByte)[1];  //** B1 reçoit 10

  TArrayReal(PTByte)[1]:=345.67;  //*** Comme une zone de réels
  R1:=TArrayReal(PTByte)[1];  //** R1 Reçoit 345.67
```

Récupération de l'adresse des procédures et fonctions

Les adresse de procédures et de fonctions sont stockées dans des pointeurs systèmes. Lorsque le programme appelle une fonction, il fait un appel indirect c'est-à-dire qui appelle l'adresse contenue dans la variable affectée à la procédure ou fonction (son pointeur).
Exemple :
Variable pointeur *STRPAS* contient l'adresse finale de la procédure STRPAS (de sysplc).
Variable pointeur *SETFILLCOLOR* contient l'adresse finale de la procédure SETFILLCOLOR (du système).

En écrivant

```
Var
 PTF,PTG : Pointer;
Begin
  PTF := ADDR (SETFILLCOLOR);
  PTG := ADDR (SETCOLOR);
  PTG^:=PTF^;  //*** Le pointer sur SETCOLOR est modifié avec l'adresse de SetFillColor
  SetColor(ClREd);  //*** Appel de SetFillColor et non plus SetColor.
End;
```

PTF Contient l'adresse de la variable Pointant sur la procédure SETFILLCOLOR.
PTG Contient l'adresse de la variable Pointant sur la procédure SETCOLOR.
PTF^ renvoie l'adresse mémoire du début de la procédure SETFILLCOLOR.

☞ **Note** : Voir aussi chapitre précédent *Chargement déchargement manuel d'une DLL*.

La librairie Windows contient la procedure ModifyAddr:

```
Procedure ModifyAddr( Var AddrDest; Var AddrSource );
```

Permettant d'échanger l'adresse du procedure ou fonction avec une autre.

Exemple :

```
Begin
  PTF := ADDR (SETFILLCOLOR);
  PTG := ADDR (SETCOLOR);
  ModifyAddr (PTG,PTF);  //*** correspond à PTG^:=PTF^;
  SetColor(ClREd);      //*** Appel de SetFillColor et non plus SetColor.
End;
```

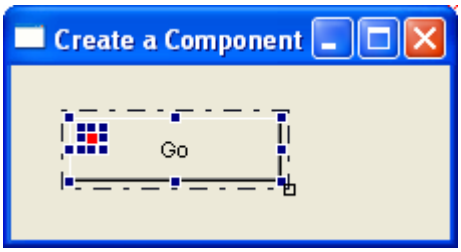
Création de composants Visual I/O

Visual I/O permet de créer sous l'éditeur des composants qui seront enregistrés dans la boite à outils.
Comment créer un composant :

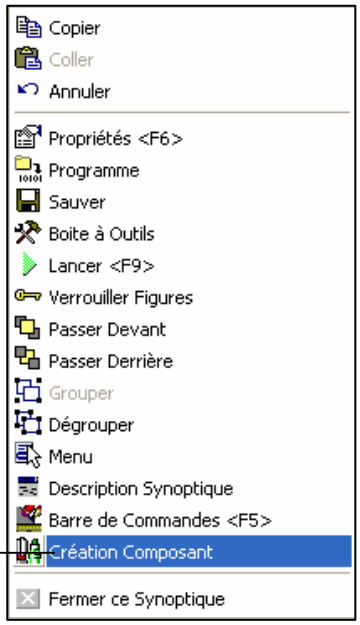
1 Créez le graphique de votre nouveau composant comme si c'était une partie d'un synoptique.
2 Placez derrière les figures les programmes d'animations ou de traitement souhaités.
3 Lancez le synoptique pour un test final.

A ce point la partie graphique et programme est débuggée. Vous pouvez alors transporter le code vers l'éditeur de code des composants Visual I/O.

Exemple de composant - ButtonLed

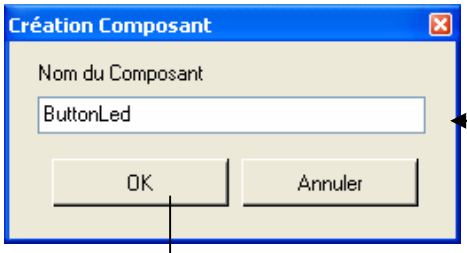


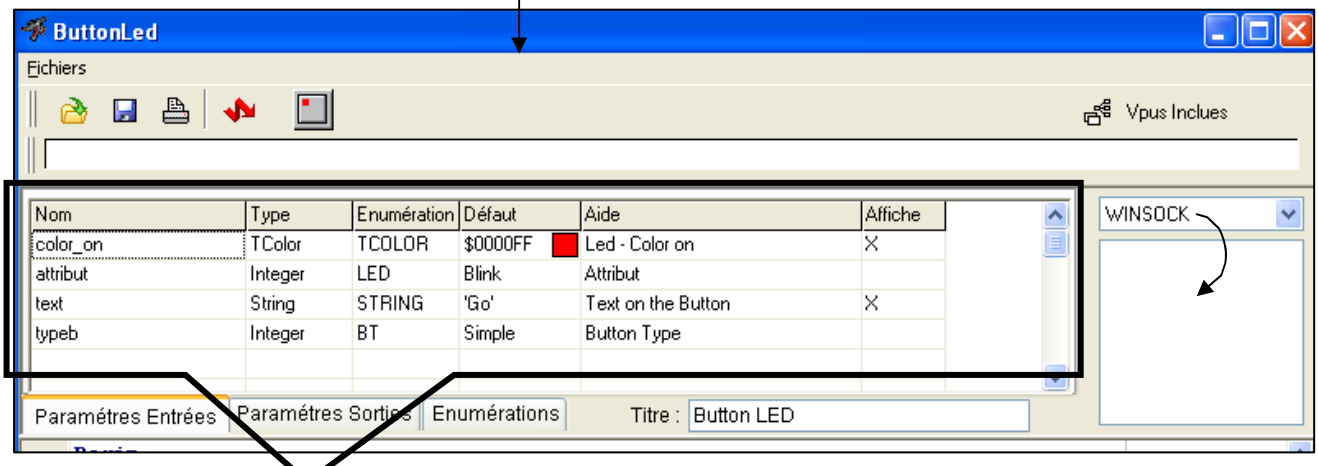
◄ *Après avoir dessiné le composant* ***groupez le****.*
Celui-ci est composé d'un bouton et dans rectangle symbolisant la Led



Cliquez avec le bouton droit de la souris,
un menu flottant apparaît.
Cliquez sur *Création Composant*

Une boite de dialogue apparaît Frappez le nom de sauvegarde du composant.
Ici ***ButtonLed*** *permet de créer un fichier compilé* ***ButtonLed.VPU.***





La partie haute permet la définition des paramètres que devra renseigner le programmeur lorsqu'il utilisera le composant dans les synoptiques.

La listebox de droite permet de lister les librairies nécessaires au programme du composant. Amenez par Drag & Drop le nom de la librairie dans le Combobox vers la listebox du dessous.

La **Zone titre** reçoit l'intitulé du composant qui apparaîtra dans la boite à outils quand la souris survolera le composant.

Paramétres Entrées | Paramétres Sorties | Enumérations | Titre : Button LED

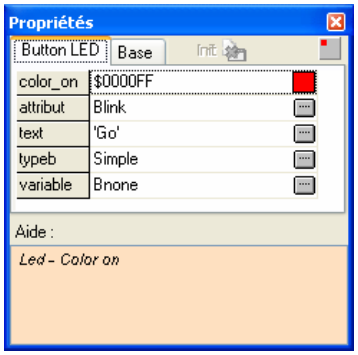
```
Begin
  If Start_Visu Then SetText(Text);
  If TypeB = 1 Then ToggleButton(Variable,-2)
               Else If Button then Variable:=Not(Variable);
  If Attribut=1 then
  Begin
    If BT_500=1 Then
    Begin
      Next_Group;
      If Variable Then
      Begin
        If GetFillcolor=Color_On Then 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;
```

Sources fournies

La partie basse reçoit le code source appliqué au premier tracé effectué sur l'écran. Afin de pouvoir piloter les autres tracés comme le rectangle de la led, la fonction **Next_Group** sera largement utilisée dans la création de composants.

Définition des paramètres utilisateur

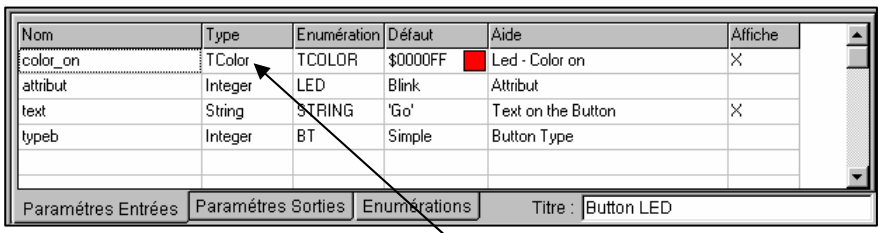
Cette fenêtre de propriétés apparaît en design après avoir placé un nouveau composant sur le synoptique



Ici 5 paramètres sont définis:

Color_On : Permet de choisir la couleur de la led. En double-cliquant sur la cellule une palette apparaît.
Attribut : Permet d'afficher une liste de possibilités (Blink, Normal).
Text : Permet d'affecter une variable (string) ou un texte défini au texte du bouton.
TypeB : Permet de choisir dans une liste prédéfinie si c'est un bouton poussoir simple ou à accrochage.
Variable : permet d'affecter une variable de type booléen à l'état du

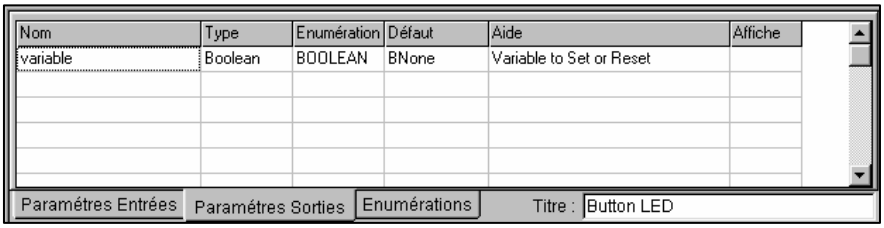
Variables et constantes en paramètres d'entrée



Frappez le nom de la variable, puis double-cliquez dans la colonne type pour faire défiler les **types** prédéfinis. Choisissez une valeur par défaut. Frappez un texte d'aide si nécessaire

Color_On est du type TColor ce qui a pour effet de permettre l'appel d'une palette dans les propriétés.
Attribut est du type Integer. Mais pour que le programmeur ne puisse choisir que des valeurs par défaut, Une énumération est stipulée (Ici Enumération de type Led voir ci après).
Text est une variable simple de type String ou une constante de défaut 'GO'.
TypeB est du type Integer et est associé à une énumération de type BT.

Variables uniquement en paramètres de Sortie



Ici le paramètre variable ne pourra recevoir qu'une variable booléenne existante dans la liste des variables globales. Par précaution la variable **BNone** est inscrite par défaut.
Cette variable est une variable Système existante qui permet de constituer des affectations par défaut.

Les énumérations

Déclarez vos énumérations dans le troisième onglet de l'éditeur.

Les énumérations permettent de prédéfinir des valeurs et d'y affecter des textes en clair.
Exemple :
La syntaxe est le mot Réservé **ENUM** suivi du nom de cette énumération.
Les énumérations (Nom : Valeur) et pour finir le mot Réservé **END** (sans ;)

```
Enum LED
  Blink  : 1  // Ici les textes Blink et Normal apparaîtrons devant la variable Attribut
  Normal : 2
End
Enum BT
  Toggle : 1   // Ici les textes Toggle et Simple apparaîtrons devant la variable TypeB
  Simple : 2
End
```

Le Programme de ButtonLed (sources fournies)

```
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;
```

Grouper les figures dans un composant

L'entrée dans le programme fait toujours référence à la première figure tracée.
Pour inverser l'ordre des figures utilisez **passer devant** et **passer derrière** dans l'éditeur.
Pour travailler au niveau programme sur d'autres figures du composant utilisez **Next_Group** ou **Previous_Group**.
Une autre fonction utile est TypeFig qui renvoie le type de figure (CLINE,CBMP etc..) pour connaître les valeurs retournées par cette fonction voir les constantes définies dans Windows.PAS.

Passer d'une figure à l'autre dans un composant.

La procédure Commut_Fig ne peut être utilisée sans connaître le nom exact des figures sur lesquelles on doit intervenir par programme.

Considérons qu'un composant contienne 3 Cercles, à chaque utilisation d'un composant dans un synoptique, le système créé 3 nouveaux cercles avec 3 noms distincts (CircleX, CircleX+1, CircleX+2). Nous ne pouvons alors pas coder en 'Dur' dans le programme du composant les noms des cercles.
Cependant il est possible de connaître en Run le nom de ces cercles dans un composant.

Exemple :
Chercher à l'init le nom des 3 cercles, puis travailler avec l'instruction commut_fig pour passer d'un cercle à l'autre.

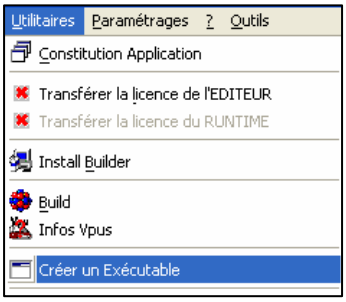
```
Var
  Name1,Name2,Name3 : String ;
Begin
  If Start_Visu Then {** mise en mémoire nom fig *
  Begin
    Name1 :=GetFigname; //* Récupère le nom du 1er Cercle
    Next_Group ;
    Name2 :=GetFigname; //* Récupère le nom du 2e Cercle
    Next_Group ;
    Name3 :=GetFigname; //* Récupère le nom du 3e Cercle
  End;
  Commut_Fig('Name2') ; //* passe sur 2e cercle
  SetFillColor(Color2);
  Commut_Fig('Name3') ; //* passe sur 3e cercle
  SetFillColor(Color3);
  Commut_Fig('Name1') ; //* passe sur 1e cercle
  SetFillColor(Color1);
End;
```

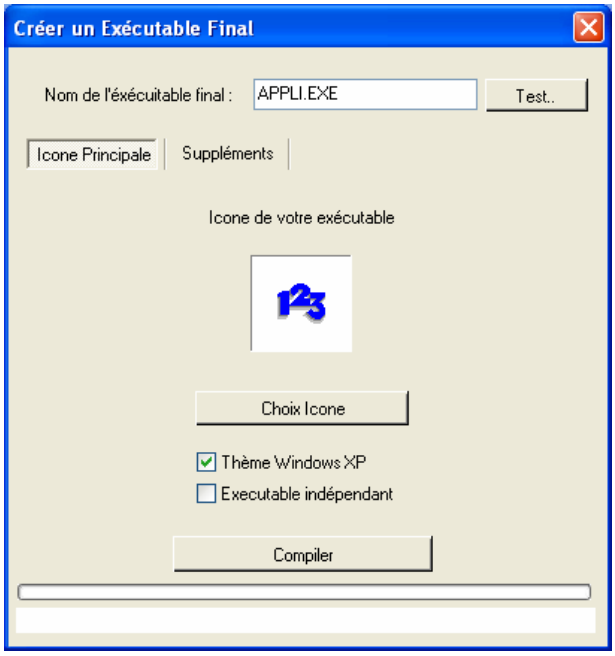
Créer un exécutable final

Visual I/O 7 contrairement aux versions antérieures, crée des exécutables incluant toutes les données nécessaires (Bitmap, menu, fichiers de configuration..) à vos applications.

Le programme généré à chaque lancement de l'application par le bouton ou par la touche **<F9>** est nommé par défaut **APPLI.EXE**.

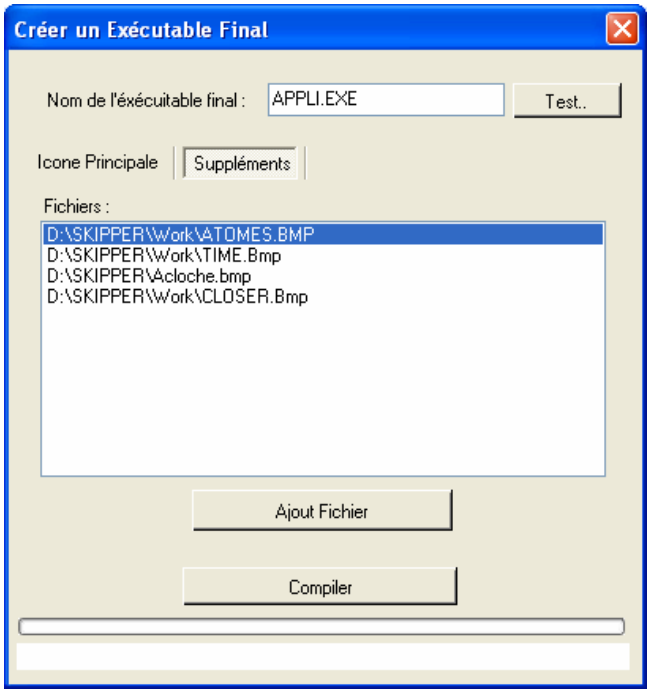
Ce nom ainsi que différents paramètres sont modifiables dans le menus utilitaires et dans la rubrique créer un exécutable





Nom de l'exécutable final Permet de spécifier le nom de l'exécutable généré à chaque compilation ou lancement de l'application depuis l'éditeur.
Bouton choix icône Permet de naviguer dans votre PC pour choisir une icône pour votre application.
Thème Windows XP Permet si vous êtes sous windows XP et dans propriété de l'affichage (dans windows) est réglé thème Windows XP d'afficher les contrôles windows avec le look XP..
Exécutable indépendant : non utilisé.
Compiler Crée le programme final APPLI.EXE.

Onglet Suppléments



Fichiers
Contient la liste des fichiers qui doivent être ajouter à l'exécutable (dans les ressources) en plus des fichiers traditionnels comme les synoptiques et autres bitmaps utilisés.

On peut trouver des fichiers tel que les fichiers HTML ou PDF par exemple.

Bouton Ajout Fichier permet d'appeler un navigateur pour le choix et l'ajout d'un nouveau fichier.

Le programme ***SCANRESEXE.exe*** permet de visualiser les ressources dans l'exécutable généré par Visual I/O.
Voir Menu *Outils ou mise à jour web*

La fonction ***ExtractRes*** permet par programme d'extraire un fichier des ressources de l'exécutable.

Debugger un programme

Quel que soit le soin apporté à l'écriture de votre programme, il contiendra toujours quelques erreurs, ou bogues, qui l'empêcheront de fonctionner comme prévu. Le débogage consiste à rechercher ces erreurs et à les réparer. L'EDI fournit un débogueur intégré qui vous aide à détecter et à corriger les erreurs contenues dans vos programmes. C'est un outil complet qui vous permet de :

- Contrôler l'exécution du programme
- Examiner la valeur des variables et des éléments des structures de données
- Modifier les valeurs des données au cours même du débogage

Pour pouvoir débugger un programme il faut le lancer en mode debug. Le programme est sous le contrôle de l'éditeur mais par ralenti pour autant puisque le code compilé reste le même sauf un échange d'octets aux points d'arrêts.

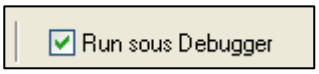
Etape 1 – Les points d'arrêts

Compiler le programme ou la librairie souhaitée en debug. Placez des points d'arrêts aux lignes souhaitées

```
{*** 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;
```

Ici un point d'arrêt est placé sur la ligne qui ouvre une connexion Tcp/IP. Tant que l'opérateur n'a pas cliqué sur le bouton, le programme ne passe pas dans le bloc Begin End. Lorsque le programme passera sur l'instruction SocketOpenConnection, le programme sera arrêté et la main donné à l'éditeur pouvant ainsi faire avancer le programme en pas à pas et visualiser les différentes variables du programme. Pour continuer le programme frappez la touche <F9>

Etape 2 – Lancer sous debug

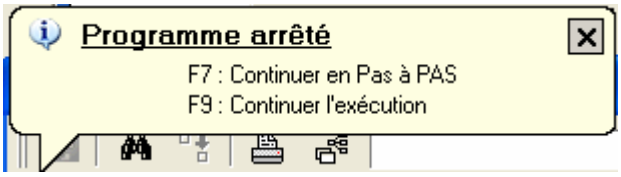


Lancez le programme de l'éditeur avec la case à cocher Run sous debugger cochée.

Le programme est compilé comme d'habitude sans aucune modification. Seulement ce programme est lancé sous un mode spécifique dans Windows C'est lc mode "Debug" qui permet de placer des points d'arrêts et de visualiser les variablesi.

Etape 3 – Arrêt du programme au point d'arrêt

Quand le programme arrive sur un point d'arrêt, il passe la main à l'éditeur en affichant une petite fenêtre (la première fois) puis en surlignant totalement la ligne de code en Rouge.



Une fenêtre indique l'éditeur ou se trouve le point d'arrêt.
Afin d'avancer en Pas à Pas frappez **<F7>**
Pour continuer normalement frappez **<F9>.**

La ligne sur laquelle le processeur c'est arrêté est surlignée.
Attention cette ligne n'est pas encore exécutée.

```
if Button Then
Begin
    TB:=SocketOpenConnection('192.168.0.1',80,DSocket);
    I:=SocketSendBuf ( DSocket , Buf, Sizeof(Buf));
    I:=SocketReceiveLength ( DSocket);
    SocketCloseConnection (DSocket);
End;
```

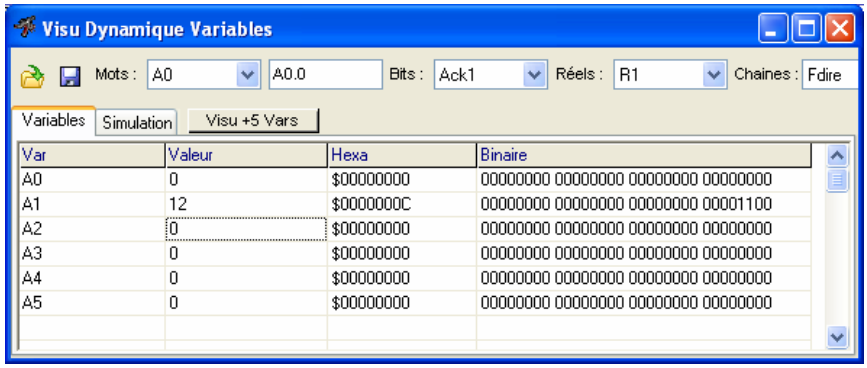
Avancez en pas à pas touche <F7>

Si un autre éditeur est ouvert et que la fonction SocketOpenConnection est visible, l'éditeur passera à la première ligne en pas à pas .Sinon le programme passe à la ligne suivante.

```
Begin
    TB:=SocketOpenConnection('192.168.0.1',80,DSocket);
    TB = True  etSendBuf ( DSocket , Buf, Sizeof(Buf));
    I:=SocketReceiveLength ( DSocket);
```

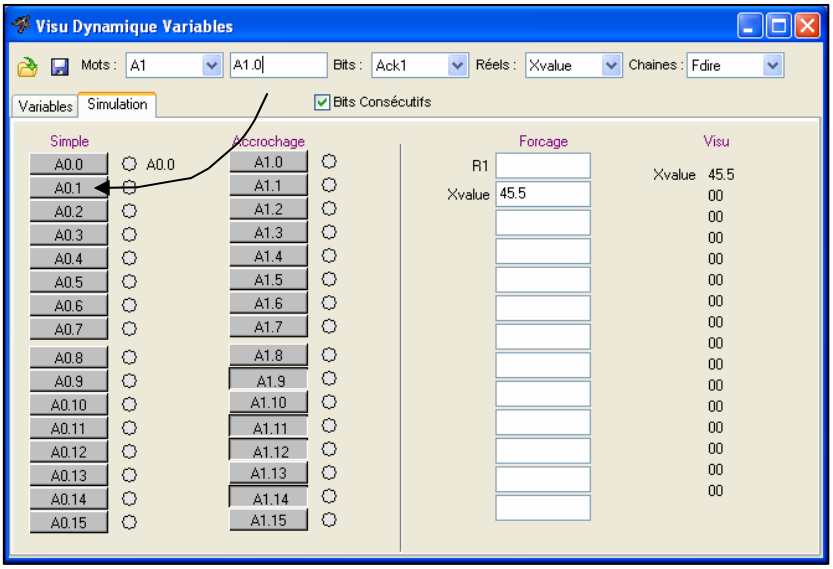
Visualisez une variable

Placez la souris sur la variable, si celle-ci est visible (et trouvée) une petite fenêtre s'affiche avec sa valeur. Ci-dessus le contenu de la variable TB est affichée.
Pour visualiser les variables globales cliquez dans le menu principal Visualisations et visudynamique.



Une grille de visualisation des variables globales uniquement est affichée. La colonne valeur, Hexa et Binaire permet de visualiser et forcer la valeur de la variable globale.

L'onglet simulation permet d'affecter des boutons poussoirs simples ou accrochage sur les variables globales booléennes.



Amenez par drag & drop la variable booléenne du combobox ou du champ saisie dans le bouton, champ de saisie ou visualisation (Visu)

La case à cocher Bits consécutifs permet de remplir automatiquement les boutons suivant avec les bits consécutifs.

Programmation avancée

Le langage Assembleur

Visual I/O permet à partir de la version 7, de programmer directement en assembleur.
Il est donc nécessaire de connaître un minimum le fonctionnement d'un processeur pour pouvoir aborder cette partie.
L'assembleur est le langage de programmation de plus bas niveau. Cela signifie qu'il est très proche du matériel. A chaque instruction correspond un code machine, qui peut être compris par le microprocesseur. Le code assembleur est donc la version lisible du code machine.
S'il est de nos jours impensable de programmer entièrement une application en assembleur (sauf à des fins d'apprentissage bien sûr), nombreuses situations font que l'on doit avoir recours à ce langage.
C'est par exemple le cas lorsque la vitesse est un facteur critique, et que la moindre microseconde est précieuse.

Exemple de programmation en assembleur

Le paramètre *range : integer* est contenu dans le registre EAX le résultat (Result) est renvoyé dans le même registre EAX.

```
{ ->EAX     Range
{ <-EAX     Result}

Var
  RandSeed : Integer; //** Variable utilisée dans la fonction random
//****  Generate a random integer
Function Random( Range : Integer) : Integer;
Asm
  Asm
     IMUL    EDX,RandSeed,08088405H  //** Multiplication de ranseed
     INC     EDX
     MOV     RandSeed,EDX     //** Récupère la variable externe Ranseed
     MUL     EDX
     MOV     EAX,EDX          //** résultat final transféré dans EAX
  End;
End;
```

Ici la fonction Random ne commence pas par Begin mais **ASM**.
Ce qui permet de ne pas affecter les registres non utilisés en entrant dans la fonction Random.

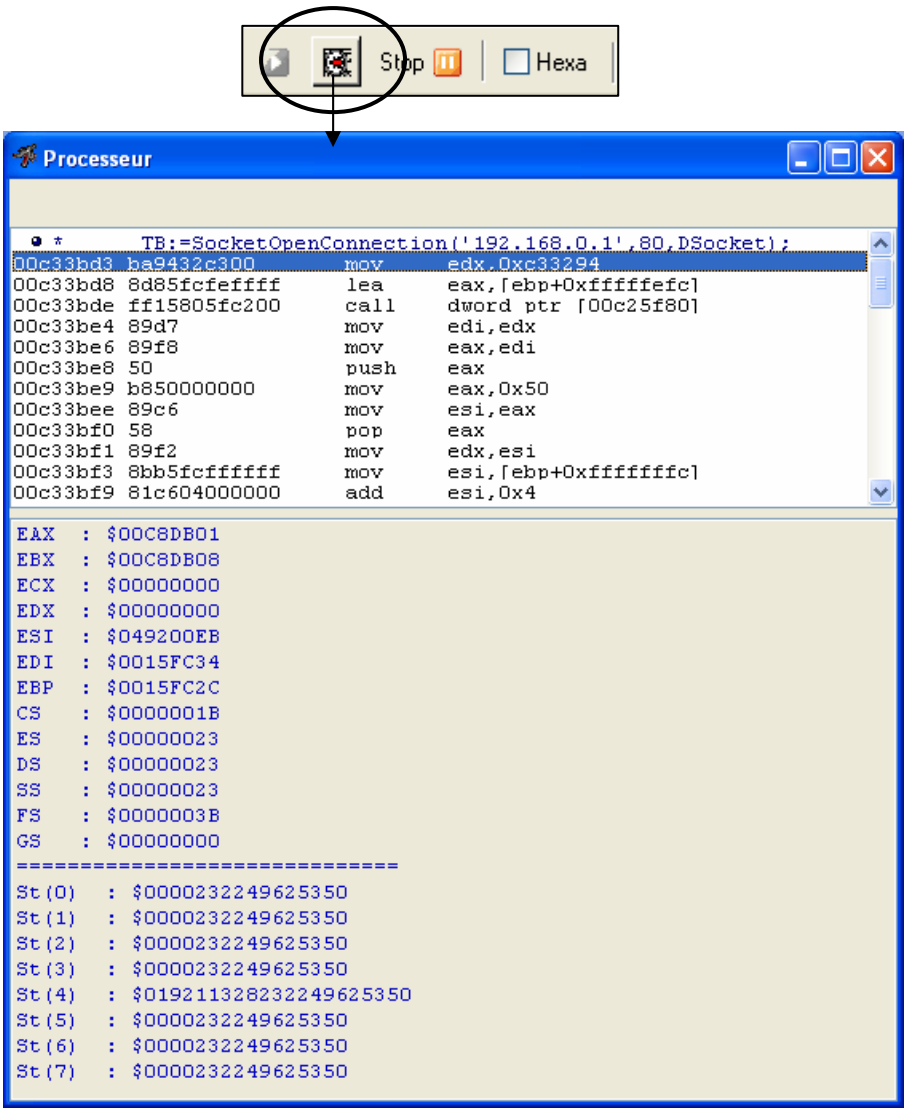
```
{** Bytes Array Comparison - Returns Address different or  0 *
Function  CompareBlockEx ( Var B1,B2 ; Nbyte : Integer) : Integer;
Begin
 Asm
   PUSH ESI
   PUSH EDI
   MOV ESI,[EAX]  //** EAX contient l'adresse de B1
   MOV EDI,[EDX]  //** EDX contient l'adresse de B2
   MOV AL,0
   CLD
   REPE CMPSB     //** REPE utilise ECX qui contient la valeur de NByte
   SETE AL
   CMP AL,1
   JZ  @OK
   MOV EAX,ESI
   JMP @FINI
@OK:              //** Etiquette OK
   MOV EAX,0
@FINI:            //** Etiquette FINI
   POP EDI
   POP ESI
 End;
End;
```

☞ **Note** : En mode *Run sous debugger*, chaque ligne d'assembleur peut être exécutée en mode pas à pas.
La visualisation des registres du processeur est aussi possible.

Visualisation des registres du processeur

En mode debug à chaque point d'arrêt ou avance en pas à pas, il est possible de visualiser les registres du processeurs et du coprocesseur arithmétique. Il est aussi possible de visualiser l'équivalent assembleur de la ligne de code en pascal. Cela démontre encore ici que visual I/O n'est pas un système interprété mais bien compilé.

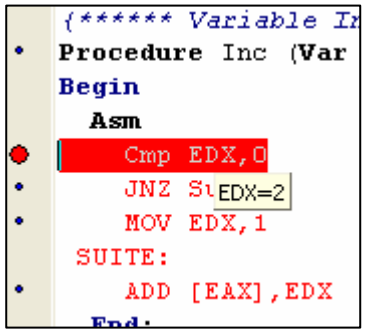
Lorsque le point d'arret est effectué, cliquez sur le bouton visu processeur.
Une fenêtre apparaît ainsi. Visualisant le code dé-assemblé ainsi que les différents Registres du processeurs ainsi que les registres du coprocesseur arithmétique.



Dans le menu principal, le bouton Stop permet aussi d'arrêter subitement l'application en cours de débuggage.
La case à cocher Hexa permet de visualiser les valeurs numériques en hexadécimal et non en décimal.

Lors d'une programmation en assembleur, il est possible de visualiser les registres du processeur en survolant le code source à la souris.

Dans le cas de la procédure **INC** de Sysplc.pas,
en mode pas à pas vous pouvez visualiser les différents registres du processeur utilisés par cette procédure.



Convention d'appel StdCall

Les procédures et fonctions de Visual I/O ont comme convention d'appel et de passage de paramètres la convention REGISTER. C'est-à-dire passage des paramètres par les registres du processeur (EAX,EDX,ECX) avec ordre de passage de Gauche à Droite.
Si la routine appelée à plus de 3 paramètres alors le passage par la pile est effectué.
Le nettoyage de la pile est fait par la routine appelée.

La convention passage **STDCALL** (plus communément utilisée dans l'api windows) à un passage de paramètres par la pile et de droite à gauche. Le nettoyage de la pile est aussi fait par la routine appelée.

La convention STDCALL est utilisé en Visual I/O dans les cas suivants.
Interface avec une DLL externe
Procédure de callback utilisée par windows.

Exemple d'utilisation

Librairie Windows.pas interface avec la DLL système User32.DLL et la fonction ShowWindow qui permet d'afficher ou de cacher une fenêtre d'un handle donné.
Ici les 2 paramètres sont passés par la pile et de droite vers la gauche (*nCmdShow* puis *hWnd*).

Function ShowWindow(**hWnd**: Integer; **nCmdShow**: Integer): Boolean; **STDCALL** 'User32.dll' name 'ShowWindow';

Librairie RICHINT.PAS contient une fonction appelée par Windows appelée StreamSave. Windows appellera cette fonction en lui passant 4 paramètres par la pile et de droite vers la gauche. Cette fonction doit restaurer le contexte de l'application visual I/O (variables) et de nettoyer la pile avant de rendre la main à 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);                //** Adresse de la function StreamSave
  EditStream.pfnCallBack := I^;  //** structure passée à windows.
End;
```

☞ **Note** : Il est aussi possible de créer des fonctions internes à visual I/O avec la convention d'appel StdCall. Celles-ci fonctionnerons tout aussi bien mais peut être moins rapidement du au mécanisme de Push et Pop des variables.

Envoie de messages Windows

Les contrôles windows fonctionnent selon la réception de messages venant d'une application.
Par exemple pour afficher ou cacher une fenêtre (Synoptique) on peut utiliser la procédure Showcontrol ou Hidecontrol mais aussi l'envoie d'un message au synoptique (fenêtre) par la fonction SendMessage et paramètre **SW_HIDE** ou **SW_SHOW.**

La fonction GetMainHwnd renvoie le handle du synoptique.
La fonction GetHWnd renvoie le handle d'un contrôle (Listbox, Combobox etc..).

```
SendMessage (GetMainHwnd, SW_HIDE,0,0);        //* pour cacher le synoptique
SendMessage (GetMainHwnd, SW_SHOW,0,0);        //* pour afficher le synoptique
```

Aide SDK windows

```
LRESULT SendMessage(
    HWND hWnd,          // handle of destination window
    UINT Msg,           // message to send
    WPARAM wParam,      // first message parameter
    LPARAM lParam };    // second message parameter
```

Contrôlez totalement Windows

Windows est basé sur l'envoie et la gestion de messages envoyés à l'application.

Pourquoi les messages ?

Les messages sont directement liés au phénomène de programmation événementielle :
Pour certaines actions, Windows génère un message. Il peut alors être récupéré par le programme qui peut ainsi réagir à cet événement.

Par exemple : vous appuyez sur le bouton gauche de la souris et un message WM_LBUTTONDOWN est automatiquement envoyé par Windows. Votre application peut être avertie de cette action de plusieurs façons.

Notamment par les événements gérés directement par le cœur du programme généré par Visual I/O qui sont déclenchés automatiquement lorsque ce message est intercepté par votre application. Vous testerez alors les actions avec les procédures disponibles (IS_ME, ARRIVE DEPART etc)

Ou par la programmation événementielle qui est donc un subtil mélange de génération de messages et réactions face à ces messages.

Principe de fonctionnement

A toute action possible sous Windows est associé un message. Ce message possède une "constante de message" qui permet de l'identifier (Voir Windows.pas ou SDK Microsoft). D'autres paramètres sont également associés au message qui permettent par exemple de récupérer la position de la souris au moment d'un click.

Structure générale d'un message tel que défini dans Windows.pas:

```
TMessage = Record
   Msg    : Integer; //** N° de message lui même
   WParam : Integer; //** 1er paramètre
   LParam : Integer; //** 2ème paramètre
   Result : Integer; //** Retour à renvoyer dans windows.
End;
```

Que se passe-t-il lors de l'émission d'un message ?

Un événement survient (cliquer sur un bouton de la souris, appuyer sur une touche..)

Cet événement est traduit par Windows sous forme d'un message de type TMessage. Windows génère un ou PLUSIEURS messages selon les cas. Par exemple dans le cas de l'appui sur une touche, Windows envoie des WM_KEYDOWN et un WM_KEYUP quand on relâche la touche (touche non système).

La plupart des messages sont alors placés dans la file d'attente des messages de l'application puis envoyés. Le cœur de l'application Visual I/O traite ces différents messages et éventuellement peut vous les transmettre.

L'application scrute en permanence cette file d'attente et agit en fonction du message qu'elle peut capter de différentes manières. Ayant capté le message de type TMESSAGE, elle a accès à tous les paramètres et donc aux renseignements la concernant. Le programme généré par Visual I/O permet aussi de vous transmettre ces messages avant même que ceux-ci soient déjà été traités par le programme. Vous pouvez alors dispatcher ou appeler des procédures spécifiques en fonction des messages reçus.

Mini gestionnaire de messages Windows

La procédure ReceiveMSG reçoit les 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;
```

La procedure **DispatchMes**, permet d'ajouter un "callback" supplémentaire de gestion des messages de Windows.

```
{*** Button1 ************************************
SObject Procedure Button1;
var
 i : Integer;
Begin
  if Start_visu Then
  Begin
    I:=Addr(ReceiveMSG);  //*** I est le pointeur sur la procReceiveMsg
    Dispatchmes(I^);   //*** Transmet le contenu du pointeur (effective adresse)
  end;
End;
```

Filtrer les messages

Grâce au handle , vous pouvez savoir si le message concerne un control particulier (listbox, combo, window, radio bouton, case à cocher, grille ..).
Par exemple Ici on teste seulement les clics dans la 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; //** récupère le handle
                                             //** de la listbox
End;
```

La librairie Bonus32

Cette librairie offre des facilités de traitement sur les chaînes de caractères.

Function Right `(S :String) : String;  //Retrieve the last character in a String;`

Renvoie le dernier caractère de la chaîne passée.
Exemple Right('ARSOFT') renvoie 'T'

Function CompareText `(S1, S2: String): Integer;  //*** Compare 2 strings`

Appelle la function CompareString de l'api windows et renvoie la valeur 0 si les 2 chaînes sont égales.
Dans cette fonction les minuscules ou majuscules sont ignorées.
```
CompareText('ARSOFT','arsoft') //renvoie 0 car les chaînes sont équivalentes.
```

Function BuildString `(StrBase : String;`
`SubStr1,SubStr2,SubStr3,SubStr4 : String) : String;`

Construit une chaîne de caractères en fonction d'un format et de paramètres.
Paramètres à insérer dans la chaîne de caractères formatée. Le < SubStr1> remplacera le caractère %1, le < SubStr2> remplacera le caractère %2, etc. Le nombre de paramètres spécifiés doit être identique ou supérieur au nombre de paramètres utilisés dans la chaîne de caractères <StrBase>. Si la chaîne de caractères <Format> contient plus de paramètres, les paramètres supplémentaires seront remplacés par des chaînes vides.
Utilisation de la fonction Buildstring dans les applications multi-langues

Si votre application contient des messages affichés dans plusieurs langues, la fonction BuildString permet de remplacer les caractères optionnels dans toutes les langues gérées.

Par exemple :
```
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;`

Formate une chaîne de caractères selon les options choisies. La Combinaison de ces options est possible.
`StrUpper, StrLower, StrBlank , Straccent .`
`Exemple :`
`FormatString('Procédure',StrUpper And StrAccent) renvoie 'PROCEDURE'`

Function StringOccurrence `( StrBase : String; Substring : String;`
`Nocasesensitive : Boolean) : Integer;`

Calcule le nombre d'occurrences d'une chaîne de caractères spécifique (en respectant les critères de recherche) dans une autre chaîne de caractères.
```
Res = StringOccurrence ('anastasia', "a");         // Renvoie 4
Res = StringOccurrence ('toto titi tito', "to");   // Renvoie 3
Res = StringOccurrence ('grille pain',  "X");      // Renvoie 0

StringOccurrence ("Le merle est dans le chêne", "le")        // renvoie 2
StringOccurrence ("Le merle est dans le chêne","le", True);  // renvoie 1
StringOccurrence ("Le merle est dans le chêne","le", False); // renvoie 3
```

Function StringComplete (StrBase : String; Size : Integer;
Complement : String) : **String**;

Renvoie une chaîne de caractères spécifique de taille déterminée. Cette chaîne peut être tronquée ou complétée par des espaces (ou par un autre caractère) pour atteindre la taille requise.

```
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;

Charge un contrôle EditBox , Listbox, combobox ou Grille de données avec le contenu d'un fichier.

Cas de la listbox et du combobox :
Ces 2 contrôles sont effacés et chargé par le contenu d'un fichier texte.
Cas de l'éditbox 'champ de saisie).
Si celui-ci est multi lignes, le chargement s'effectue comme une listbox ligne par ligne.
Si celui-ci est un champ de saisie mono ligne (editbox simple).
Seule la première ligne est affectée par la première ligne du fichier texte.

Filename : Nom du fichier Texte à charger dans le contrôle.
CtrlName : Nom du contrôle (nom de la figure graphique)
Si aucun nom la figure courante est utilisée sans ça un commut_fig est effectué.
Option : Chaîne de caractères utilisée pour passer une option (ex séparateur dans grille).

Exemple : ControlLoadFromFile('NOTICE.TXT'); Charge de la figure courante
ControlLoadFromFile('NOTICE.TXT','LISTBOX1'); Charge dans la listbox ListBox1
ControlLoadFromFile(PARAMETERS.CFG','WINDOW1',';'); Charge dans la grille Window1 avec le séparateur ; (format csv)

Function ControlSaveToFile (FileName : String; CtrlName : String;
Option : String) : Boolean;

Sauve le contenu d'un contrôle EditBox , Listbox, combobox ou Grille de données dans un fichier..

Cas de la listbox et du combobox :
Le contenu de ces 2 contrôles sont sauvegardés dans un fichier texte.
Cas de l'éditbox 'champ de saisie).
Si celui-ci est multi lignes, ltoutes les lignes sont sauvegardées dans le fichier.
Si celui-ci est un champ de saisie mono ligne (editbox simple), seule la première ligne est sauvegardée dans le fichier texte.

Filename : Nom du fichier Texte dans lequel la sauvegarde doit s'effectuer.
CtrlName : Nom du contrôle (nom de la figure graphique)
Si aucun nom la figure courante est utilisée sans ça un commut_fig est effectué.
Option : Chaîne de caractères utilisée pour passer une option (exemple le séparateur à utiliser).

Exemple : ControlSaveToFile('NOTICE.TXT'); Sauve de la figure courante
ControlSaveToFile ('NOTICE.TXT','LISTBOX1'); Sauve la listbox ListBox1
ControlSaveToFile (PARAMETERS.CFG','WINDOW1',';'); Sauve le contenu de la grille avec comme séparateur de colonne le caractère ; (création du format CSV d'Excel)

Function GetHandle (CtrlName : String) : Integer;
Renvoie le handle de fenêtre d'un contrôle windows (Window, Listbox, Combo). Le handle d'un contrôle est souvent nécessaire pour utiliser lAPI windows (Voir Richedit control par exemple).

La librairie Bonus1

Function DiskInDrive `(Drive: String): Boolean;`
Renvoie True si un disque est dans le lecteur spécifié. Aucune différence n'entre minuscule et majuscule.

Exemple : Diskindrive('a');

La librairie Gestfile

Cette librairie manage les fichiers à accès directs ou fichiers binaires. Ces instructions **ne s'appliquent pas** aux fichiers textes décrits précédemment.

```
Const
  FmRead      = $80000000;
  FmWrite     = $40000000;
  FmReadWrite = $C0000000;
```

Function FileOpen `( FileName : string; Mode: Integer): Integer;`
Ouvre un fichier en utilisant le mode d'accès spécifié. Le fichier est un fichier binaire et non texte.
La valeur du mode d'accès résulte des constantes fmXXXX.
Si la valeur renvoyée est positive, la fonction s'est bien déroulée et la valeur représente le descripteur du fichier ouvert. Si la valeur renvoyée vaut -1, cela indique qu'une erreur s'est produite.
Var
HDLE : Integer;
Exemple :

```
HDLE:=FileOpen('DATABIN.BIN', FmReadWrite);
If Hdle>0 then FileRead (Hdle,Buf,1024);
```

Function FileCreate `( FileName : string): Integer;`
crée un nouveau fichier.
La fonction FileCreate crée un nouveau fichier avec le nom spécifié. Si la valeur renvoyée est positive, la fonction s'est bien déroulée et cette valeur correspond au descripteur du nouveau fichier. Si la valeur renvoyée vaut -1, cela indique qu'une erreur s'est produite.

```
HDLE:=FileCreate('DATABIN.BIN');
If Hdle>0 then FileWrite (Hdle,Buf,1024);
```

Procedure FileClose `( Handle   : Integer);`
ferme le fichier spécifié.

```
HDLE:=FileOpen('DATABIN.BIN', FmReadWrite);
…..
If Hdle>0 then Fileclose(Hdle);
```

Function FileDelete `( FileName : String) : Boolean;`
Cette fonction efface un fichier existent et renvoie true si le fichier à été efface.

```
If Filedelete('TEST.BIN') Then Message('File deleted'); // Message dans sysplc
```

Function FileRead `(Handle : Integer; var Buffer; Count : Integer): Integer;`
lit dans un fichier le nombre d'octets spécifié. Le résultat de la fonction est le nombre d'octets lus qui ne peut être inférieur à Count
Le code suivant remplit un tampon à partir d'un fichier.

```
ActualRead := FileRead(MyFileHandle, Buffer, SizeOf(Buffer));
```

Function FileWrite `(Handle : Integer; var Buffer; Count : Integer ): Integer;`
La fonction FileWrite écrit Count octets depuis Buffer dans le fichier indiqué par Handle. La valeur renvoyée est le nombre d'octets effectivement écrits, ou -1 si une erreur s'est produite.

```
HDLE:=FileOpen('DATABIN.BIN', FmReadWrite);
If Hdle>0 then FileRead (Hdle,Buf,1024);
```

Function FileSeek `( Handle, Offset: Integer): Integer;`
positionne le pointeur d'un fichier préalablement ouvert.

```
FileSeek(MyFileHdle,FileSize(MyFileHdle));
```

Function FileSize `( Handle : Integer) : Integer;`
renvoie la taille d'un fichier (en octets).
Pour utiliser FileSize, le fichier doit être ouvert. F est une variable fichier. Si le fichier est vide, FileSize(F) renvoie 0.

```
FileSeek(MyFileHdle,FileSize(MyFileHdle));
```

Function CopyBlocksToPos `(Source,Destination : String;`
`Position :Integer) : Boolean;`

Copy une partie du fichier source du début de celui-ci jusqu'à la position spécifiée (non incluse) dans le fichier destination.

```
// Copy le début du fichier
CopyBlocksToPos (FileNameHisto,SaveFile,PosFile);
// Copy la fin du fichier
CopyBlocksFromPos(FileNameHisto,'Tempx.dbf',PosFile);
```

Function CopyBlocksFromPos `( Source,Destination : String;`
`Position : Integer) : Boolean;`

Copy une partie du fichier source à partir de la position spécifiée (incluse) jusqu'à la fin de celui-ci dans le fichier destination.

```
// Copy le début du fichier
CopyBlocksToPos (FileNameHisto,SaveFile,PosFile);
// Copy la fin du fichier
CopyBlocksFromPos(FileNameHisto,'Tempx.dbf',PosFile);
```

Traitements sur les répertoires

Function DirectoryCreate(NameDir : String) : Boolean;
Cette fonction crée un nouveau répertoire (directory) si celui-ci n'existe pas. Cette fonction renvoie true si la directory à été créée.

```
DirectoryCreate('C:\MyProj\Datas\Files');
```

Function DirectoryRemove(DirName : String) : Boolean;
Efface un répertoire (directory) si celui-ci est vide.

Function ChangeDirectory(DirName : String) : Boolean;
La fonction Changedir change le répertoire en cours en prenant en compte le chemin d'accès spécifié par DirName. Si DirName indique une lettre de lecteur, le lecteur de disque en cours est également changé.

Function GetCurrentDir : String;
Renvoie le répertoire en cours.

Function DirectoryExist (DirName: string) : Boolean;
indique si le répertoire spécifié existe. S'il existe, la fonction renvoie True. Sinon, elle renvoie False.

Function FileExtractName(FileName: string): string;
Extrait le nom et l'extension de FileName.

```
Ex : FileExtractName ('C:\API32\VPLC.EXE'); retourne VPLC.EXE
```

Function FileExtractPath(FileName: String): String;
Extrait le lecteur et le répertoire d'un nom de fichier

```
Ex : FileExtractPath ('C:\API32\VPLC.EXE'); retourne C:\API32\
```

Function FileExtractExt (FileName: string): string;
Renvoie la partie extension de FileName.

```
Ex : FileExtractExt ('C:\API32\VPLC.EXE'); retourne .EXE
```

Function ProgRunning (PName : String) : Boolean;
Retourne True si le programme spécifié est lancé sous windows. PName doit contenir le chemin complet du programme.
PNAME='APPLI.EXE' renvoie True alors que PNAME='C:\WORK\APPLI.EXE' pourrai renvoyer False malgré que ce soit le même programme spécifié.

La librairie RichInt – Interface avec un Contrôle RichEdit

Cette librairie contient toutes les procédures d'interface avec un Contrôle Richedit. Pour illustrer ou utiliser un éditeur de texte évolué chargez le modèle d'écran **Richeditmain.VPK** ou l'exemple Visual I/O.
Pour un premier aperçu se rapporter au chapitre *Richedit – éditeur de texte formaté de Windows.*

Ajoute une ligne à la suite.
Function RichAddstring `( Value : String; Handle : Integer ) : Integer;`
Ex: `RichAddstring('Visual I/O – Visual Pascal',GetHandle('Window1'));`

Efface complètement l'éditeur de texte.
Procedure RichClear `( Handle : Integer);`

Affecte un texte contenant plusieurs lignes directement dans l'éditeur.
Procedure RichSetText `( PTText : Integer; Handle : Integer );`
PTText est un pointeur sur chaîne de caractères pouvant provenir d'un buffer de lecture d'un fichier binaire par exemple :

```
FileRead(Fic,Buf[0],1024);
RichSetText(Addr(Buf[0]),HDR);
```

Renvoie le nombre de lignes d'un richedit.
Function RichGetCount `( Handle : Integer) : Integer;`

Renvoie la longueur en octets du contenu du richedit.
Function RichGetTextlen `( Handle : Integer) : Integer;`

Insert une chaîne de caractères à un endroit précis. Première ligne à l'index 0
Procedure RichInsert `( Index: Integer; S: string; Handle : Integer );`

Charge un fichier RTF ou texte dans un richedit.
Procedure RichLoadFromFile `( Filename : String; Handle : Integer);`

Sauve le contenu du richedit dans un fichier RTF.
Procedure RichSaveToFile `( Filename : String; Handle : Integer);`

Copy/Paste Commands

Efface le texte sélectionné du contrôle de saisie.
Procedure RichClearSelection `( Handle : Integer);`

Copie le texte sélectionné du contrôle de saisie dans le Presse-papiers au format CF_TEXT.
Procedure RichCopyToClipboard `( Handle : Integer);`

Copie le texte sélectionné dans le Presse-Papiers au format CF_TEXT, puis efface la sélection.
Procedure RichCutToClipboard `( Handle : Integer);`

Copie le contenu du Presse-Papiers dans le contrôle de saisie en remplaçant la sélection en cours.
Procedure RichPasteFromClipboard`( Handle : Integer);`

Annule toutes les modifications effectuées
Procedure RichUndo `( Handle : Integer);`

Procedure RichSelectAll `( Handle : Integer);`

Attributs du Texte

Autorise ou interdit la modification du texte sélectionné.
Procedure RichSetProtected `( Value: Boolean; Handle : Integer);`

Indique la couleur du texte.
Procedure RichSetColor `( Value: TColor; Handle : Integer);`

Indique la hauteur de la fonte, exprimée en points.
Procedure RichSetFontSize `( Value: Integer; Handle : Integer);`

Spécifie ou le nom de la police du texte sélectionné.
Function RichSetFontname (Value: String; Handle : Integer) : Boolean;

Détermine si la fonte est normale, italique, en gras, soulignée ou biffée.
Procedure RichSetFontStyle (Value: Integer; Handle : Integer);
Value est une des valeurs ci-dessous ou une combinaison de ces valeurs :

```
CFM_BOLD            = $00000001;
CFM_ITALIC          = $00000002;
CFM_UNDERLINE       = $00000004;
CFM_STRIKEOUT       = $00000008;
```

Demande que le texte sélectionné soit en gras et Souligné .
Ex : `RichSetFontStyle ( CFM_BOLD Or CFM_UNDERLINE, HDR);`

Renvoie les attributs de la fonte du texte sélectionné.
Renvoie une des valeurs ci-dessus ou une combinaison de ces valeurs.
Function RichGetFontAttributes (Handle : Integer) : Integer;

Alignements Attributs

Procedure RichSetAlignment (Value: TAlignment; Handle : Integer);
Fixe l'alignement du texte sélectionné. Value peut prendre une des 3 Valeurs :

```
taLeftJustify       = 0; //** Texte aligné à Gauche
taRightJustify      = 1; //** Texte aligné à Droite
taCenter            = 2; //** Texte Centré
```

Spécifie si le paragraphe sélectionné fait partie d'une énumération. Pour insérer une puce à gauche du paragraphe. Value peut prendre les valeurs suivante :
Procedure RichSetNumbering (Value: TNumberingStyle; Handle : Integer);
nsNone Pas de puces à gauche.
nsBullet Des puces sont placées à gauche avec un décalage du paragraphe suffisant pour les puces.

Renvoie l'information de presence de puces sur le texte sélectionné. La valeur renvoyée est une des 2 valeurs ci-dessus (nsNone ou nsBullet).
Function RichGetNumbering (Handle : Integer) : TNumberingStyle;

Spécifie le décalage, exprimé en pixels, de la première ligne du paragraphe par rapport à la marge gauche.
Procedure RichSetFirstIndent (Value: Integer; Handle : Integer);

Renvoie le décalage, exprimé en pixels, de la première ligne du paragraphe par rapport à la marge gauche.
Function RichGetFirstIndent (Handle : Integer) : Integer;

Renvoie le décalage, exprimé en pixels, du paragraphe par rapport à la marge gauche.
Function RichGetLeftIndent (Handle : Integer) : Integer;

Spécifie le décalage, exprimé en pixels, du paragraphe par rapport à la marge gauche.
Procedure RichSetLeftIndent (Value: Integer; Handle : Integer);

Chercher

Recherche une chaîne dans une plage donnée du Richedit.

```
Function RichFindText ( SearchStr: string;  StartPos, Len: Integer;
                        Options: TSearchTypes; Handle : Integer): Integer;
```

Utilisez la méthode FindText pour rechercher dans la chaîne de texte d'un contrôle de éditeur de texte formaté la chaîne spécifiée par le paramètre SearchStr. La recherche ne porte que sur la partie du texte commençant à la position StartPos et de longueur Len. Utilisez la paramètre Options pour spécifier si la recherche ne doit chercher que des mots entiers ou si la recherche doit tenir compte des différences majuscules/minuscules. RichFindText renvoie la position de la chaîne recherchée où 0 indique le premier caractère du contrôle éditeur de texte formaté (et non pas le premier caractère de la zone de recherche). Si la chaîne recherchée n'apparaît pas dans l'intervalle spécifié, RichFindText renvoie -1.
L'utilisation de RichFindText au lieu de rechercher par code dans le texte évite les problèmes liés au codage des caractéristiques du texte formaté. Options peut prendre une des valeurs suivantes :

```
stWholeWord = 0;  //** tous les mots
stMatchCase = 1;  //** Mot exact différence minuscule et majuscule.
```

La librairie Sysplc

Cette librairie s'appuie sur l'API windows (Procédure et fonctions disponible dans les DLL du système de Windows). Toutes les procédures ne seront pas détaillées seules les plus intéressantes sont décrites.

Joue un son waveform équivalent à SystemHand.

Procedure Beep;

Affiche une petite fenêtre avec le message contenu dans texte. Cette fenêtre n'à pas de titre.

Procedure Message `(Texte : String);`

Affiche une petite fenêtre avec le message contenu dans texte et un titre contenu dans Caption.

Procedure MessageEx `( Texte,Caption : String);`

Affiche une fenêtre avec le texte et les 2 boutons OUI NON. Renvoie True si le bouton OUI est cliqué.

Procedure MessageYesNo `( Texte : String):Boolean;`

Joue un fichier son.

Procedure Sound `( SoundFile : String);`

```
Ex : Sound(C:\WINDOWS\Media\notify.wav);
```

Inverse le poids fort et poids faible d'un Word ou d'un integer (mais que les 16 premiers Bits);

Procedure XChg16 `( Var V : Integer);`

```
Var
  V : Word; ou V : Integer;
V:=$FF11;
Xchg16(V);  //** V est alors égal à $11FF
```

StrPas convertit PSTR qui est une chaîne terminée par le caractère Null, en une chaîne Pascal.

Function StrPas `( Var PStr ) : String;`

Le paramètre PStr est en fait un pointeur sur chaîne de caractères style chaîne de caractère en langage C.

Ces 2 fonctions compare 2 zones d'octets démarrant à l'adresse B1 et B2 . La comparaison s'effectue sur un nombreux d'octets définis dans la variable NByte. *CompareBlock* renvoie true si les 2 zones sont équivalentes et *CompareBlockEX* revoie 0 si les 2 zones sont équivalentes sinon l'adresse différentes détectée est renvoyée.

Function CompareBlock `( Var B1,B2;  Nbyte : Integer) : Boolean;`

Function CompareBlockEx `( Var B1,B2 ; Nbyte : Integer)  : Integer;`

La fonction GetTick retourne le nombre de millisecondes déjà écoulées depuis le démarrage de Windows.
The GetTick function retrieves the number of milliseconds that have elapsed since Windows was started.

Function GetTick `: Integer;`

Transforme une chaîne Pascal (VisualI/O) en pointeur sur chaîne de caractères compatible avec l'API windows ou autres langage (Langage C par exemple).

Function Pchar `( Var Str : String) : Integer;`

Function Random `( Range : Integer) : Integer;`

Function RandomFloat `( Range: integer)  : Real;`

Ces 2 fonctions renvoient un nombre aléatoire entier ou à virgule flottante.
La fonction Random renvoie un nombre aléatoire compris dans l'étendue 0 <= X < Range.
Random génère des nombres entiers aléatoires dans une étendue spécifiée.
RandomFloat génère des nombres à virgule flottante (Réels) aléatoires dans une étendue spécifiée.

Function DialogBox `( Caption, Texte : String; Utype : Integer) : Integer;`

Affiche une boite de dialogue de confirmation avec une barre de titre (Caption) et un texte (Texte).
Utype est le type de boite de dialogue à afficher .
Voir les constante dans le source .

```
MB_OK  MB_OKCANCEL  MB_ABORTRETRYIGNORE  MB_YESNOCANCEL MB_YESNO MB_RETRYCANCEL
//******* Icônes combinables *******
MB_ICONHAND, MB_ICONQUESTION, MB_ICONEXCLAMATION, MB_ICONASTERISK, MB_USERICON,
```

Le retour de cette fonction en en relation avec l'affichage demandé :

```
IDABORT ou IDCANCEL ou IDIGNORE ou IDNO ou IDOK ou IDRETRY ou IDYES
```

Librairie Windows

Cette librairie s'appuie sur l'API windows (Procédure et fonctions disponible dans les DLL du système de Windows).
Nous rentrons ici dans l'interface directe des procédures et fonctions de l'API windows (Voir SDK Microsoft).
.

Function Open_File `(Var NomFichier : String;`
`TypeFichier, Filtre, Titre : String) : Boolean;`
Function Save_File `(Var NomFichier : String;`
`TypeFichier, Filtre, Titre : String) : Boolean;`
Affiche une boîte de dialogue permettant la sélection d'un fichier à sauvegarder.
Affiche une boîte de dialogue permettant la sélection d'un fichier à charger.
.

Procedure Set_Focus `(Control : String);`
Permet de passer le Focus à une figure précise. Cette figure est de type contrôle Windows Listbox, Combobox, Case à cocher, Radio bouton Grille, fenêtre et dérivé. Le focus est donné quand le curseur permet la saisie .

Renvoie true si le synoptique à le focus (barre de titre active).
Function GetWinFocus `: Boolean;`

Permet de retracer entièrement tous les éléments graphiques sur le synoptique.
Procedure FormRefresh;

Function ShellExecute `( hWnd: Integer; Operation, FileName, Parameters,`
`Directory : PAnsiChar; ShowCmd: Integer): HINST;`
Permet d'ouvrir ou imprimer un fichier spécifique en spécifiant le nom de l'exécutable qui doit le recevoir.
Paramètre **Operation** chaîne de caractères

'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 opens the file, as if "open" had been specified.

'explore' The function explores the folder specified by FileName.

ShowCmd voir constantes Windows.pas (SW_HIDE, SW_MAXIMIZE, SW_MINIMIZE Etc..)

Exemple ci-dessous
Ouvrir Excel normalement avec le fichier Histo.dbf. l'intérêt de cette instruction réside dans le fait que vous n'êtes pas obligé de connaître le chemin complet d'excel.

```
ShellExecute(0, PChar('Open'), PChar('excel.exe'), Pchar('Histo.dbf'), nil,
SW_SHOWNORMAL);
```

Pour explorer le répertoire C:\Windows utilisez cet appel suivant :

```
ShellExecute(0, Pchar('explore'), Pchar('C:\WINDOWS'), 0, 0, SW_SHOWNORMAL);
```


Pour les Autres fonctions et procédures se référer à l'API Windows.

MultiTaches et Threads

L'expression applications Multithreads désigne communément une application qui exécute plusieurs programmes en parallèle.
Quand une application (Programme) est exécutée, elle est chargée en mémoire, prête à s'exécuter. A ce moment, elle devient un processus comportant un ou plusieurs threads contenant les données, le code et d'autres ressources système du programme. Un thread exécute une partie d'une application et se voit attribué du temps CPU par le système d'exploitation. Tous les threads d'un processus partagent le même espace d'adresse et peuvent accéder aux variables globales du processus.

L'utilisation des threads améliore les performances d'une application en :
Gérant les entrées de plusieurs périphériques de communication.
En distinguant des tâches de priorités différentes. Par exemple, un thread de priorité élevée gère des tâches critiques alors qu'un thread de faible priorité gère les autres tâches.

L'utilisation des threads doit se faire en respectant les recommandations suivantes:

Conserver la trace d'un trop grand nombre de threads gaspille du temps CPU; sur un système disposant d'un seul processeur, la limite est de 16 threads par processus.
Si plusieurs threads actualisent les mêmes ressources, synchronisez les threads pour éviter les conflits.

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;

Exemple
Créer 2 taches indépendantes et parallèles au programme principal.

```
//*** Tache 1 ***************
Procedure Task1;
Begin
  Repeat
    A0:=A0+1;
    Sleep(10); //** passe la main au système d'exploitation
  Until False;
End;
//*** Tache 2 ***************
Procedure Task2;
Begin
  Repeat
    A1:=(A1*2)+10;
    Sleep(20);  //** passe la main au système d'exploitation
  Until False;
End;

{*** Button1 ***********************}
SObject Procedure XPBUTTON1;
Var
  ID1,ID2 : Integer;
Begin
 If Start_visu Then
 Begin
   ID1:=Thread (Addr(Task1),6); //** Démarre Tache 1
   ID2:=Thread (Addr(Task2),6); //** Démarre Tache 2
 End;
 If Button then
 Begin
   Destroythread(ID1); //** Détruit Tache 1
   Destroythread(ID2); //** Détruit Tache 2
 end;
End;
```

Function Thread (Proc : Integer; Priority : Integer) : Integer;
Proc : Adresse de la procédure à exécuter par le thread.
Priority : Priorité d'exécution (Détermine le niveau de priorité du thread relativement aux autres threads du processus.)

0 Le thread ne s'exécute que lorsque le système est inoccupé.
Windows n'interrompt pas les autres threads pour exécuter un thread de priorité tpIdle.
1 La priorité du thread est deux points en dessous de la normale.
2 La priorité du thread est un point en dessous de la normale.
3 La priorité du thread est normale.
4 La priorité du thread est un point au dessus de la normale
5 La priorité du thread est deux points au dessus de la normale.
6 Le thread a la priorité la plus élevée.

Retour : Handle du thread créé.

Function DestroyThread (hThread : THandle) : Boolean;
Arrête le thread et le détruit complètement. Il faudra recréé le thread par la fonction Thread.

Procedure Sleep (dwMilliseconds : Integer);
La fonction suspend l'exécution du thread courant pour un intervalle de temps spécifié et permet au système d'exploitation d'exécuter un autre thread donc une procédure.
La procédure Task1 de thread est une boucle infinie. Cette tache ne bloque pas le processeur car elle est exécutée par un thread. De plus cette tache rend la main au système d'exploitation pendant 10ms.

```
Procedure Task1;
Begin
  Repeat
    A0:=A0+1;
    Sleep(10); //** passe la main au système d'exploitation
  Until False;
End;
```

Function ResumeThread (hThread : THandle) : Integer;
Function SuspendThread (hThread : THandle): Integer;

Ces 2 fonctions permettent de suspendre et de redémarrer l'exécution d'un thread. Chaque appel de suspendthread incrémente un compteur interne et resumethread décrémente. Lorsque le compteur est égal = 0 alors le thread redémarre.

Function SetThreadPriority (hThread: THandle; nPriority: Integer): Boolean;
Détermine de nouveau le niveau de priorité du thread relativement aux autres threads du processus. Cette fonction n'est généralement pas nécessaire à appeler car la priorité est fixée par la fonction Thread.

Attention : "Gonfler" la priorité du thread pour une opération utilisant intensivement la CPU peut "sous-alimenter" les autres threads de l'application. Il ne faut accorder une priorité élevée qu'à des threads qui passent l'essentiel du temps à attendre des événements extérieurs.

Exécution de programme sur événements clavier ou souris

La variable **EventAPP** permet de savoir si un évènement clavier ou souris c'est produit dans le synoptique. Grâce à cette variable vous pouvez conditionner des actions ou des traitements si seulement un click souris ou clavier est effectué par l'opérateur.

Exemple :

```
{*** Value1 ************************************}
SObject Procedure Value1;
Begin
  If EventAPP Then //** calcul effectué si évènement souris ou clavier
  Begin
    R1:= Exp(R2) / 2;
    R1:= R1 + R2 / R1;
  End;
End;
```

Interdire la fermeture d'un synoptique

La procédure **SetCanclose** permet d'autoriser ou interdire la fermeture du synoptique par son bouton localisé dans la barre de titre.

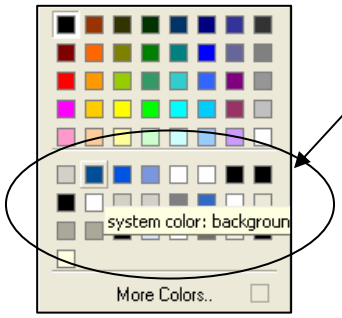
```
{*** Button1 ************************************}
SObject Procedure XPBUTTON1;
Begin
 SetCanClose(False); //** Interdit la fermeture du synoptique
 If Windisparition Then Message('Ciao'); //** Détecte la fermeture du synoptique
 If Button then CLoseAppli; //** Quitte le synoptique par programme
End;
```

Couleurs système

Afin de désigner une couleur il existe dans Visual I/O le type TColor qui est fait un integer. Plusieurs types comme Tcolor sont des integers (THandle, Dword etc..).
Une variable TColor représente une couleur :
Si vous spécifiez une valeur TColor en tant que valeur hexadécimale de 4 octets au lieu d'utiliser les constantes définies dans l'unité Windows, les trois premiers octets représentent respectivement l'intensité RGB des couleurs bleu, vert et rouge. La valeur 00FF0000 représente un bleu pur de pleine intensité, 0000FF00, un vert pur de pleine intensité et 000000FF, un rouge pur de pleine intensité. 00000000 représente le noir et 00FFFFFF, le blanc.

L'unité Windows contient la définition de constantes TColor utiles. Ces constantes renvoient soit directement à la couleur la plus approchante dans la palette système (par exemple, clBlue renvoie au bleu), soit à la couleur de l'élément écran du système telle qu'elle est définie dans la section Couleur du Panneau de configuration Windows (par exemple, clBtnFace renvoie à la couleur système des faces de boutons).

Si l'octet de poids fort vaut zéro (00), la couleur obtenue est celle la plus approchante dans la palette système. Si l'octet de poids fort vaut un (01), la couleur obtenue est celle la plus approchante dans la palette actuellement réalisée. Si l'octet de poids fort vaut deux (02), la couleur obtenue est celle la plus approchante dans la palette logique du contexte de périphérique en cours.



La palette de couleur de l'éditeur permet de choisir une des couleurs système de windows. Choisir une des couleurs système permet de suivre la définition de celles-ci lorsque vous changez de thème dans le panneau de configuration de Windows.
Par exemple la couleur Button Face qui règle la couleur d'un bouton est définie gris clair mais avec des nuances de gris entre Windows classique et Windows XP. En choisissant cette couleur système la couleur de vos boutons suivrons automatiquement le changement de Thème.

Function ColorToRGB(Color: TColor): Integer; *//** convertit une valeur TColor en sa représentation RGB*
Function GetSysColor(nIndex: Integer): TColor; *// Retrieves the current color of the specified display element*

Les fichiers INI

La VPU **INIFILES** est une enveloppe de bas niveau pour le système de fichiers INI Windows 3.x 16 bits, qui autorise le stockage et la lecture d'informations et de paramètres propres à une application dans un fichier INI.

Le format texte des fichiers INI est le moyen standard utilisé par les applications Windows 3.x pour stocker et lire des paramètres applicatifs d'une session à l'autre. Un fichier INI stocke des informations par groupes logiques appelés "sections". Le fichier WIN.INI, par exemple, contient une section appelée "[Desktop]". Dans chaque section, les valeurs de données réelles sont stockées dans des clés nommées. Ces clés sont de la forme :

<keyname>=<value>

Le FileName transmis à chaque appel de fonction est le nom du fichier INI.

Liste des Fonctions de d'écriture et de lecture.

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;

Exemple d'écriture dans un fichier INI (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;
```

Résultat dans le fichier Fungame.ini:

```
[Software]
Name 1=Visual I/O
Name 2=Visual PLC
[Options]
Level=3
Sound=1
[Configuration]
ShowToolBar=1
```

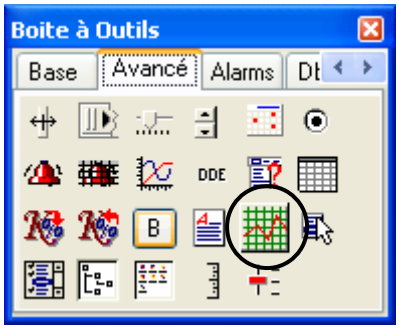
Cas de la lecture exemple

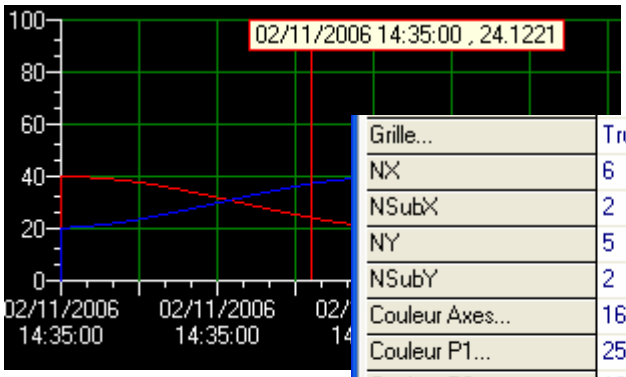
```
    TS:= IniFileReadString ('FUNGAME.INI','Software', 'Name 1','ARSOFT Visual');
```

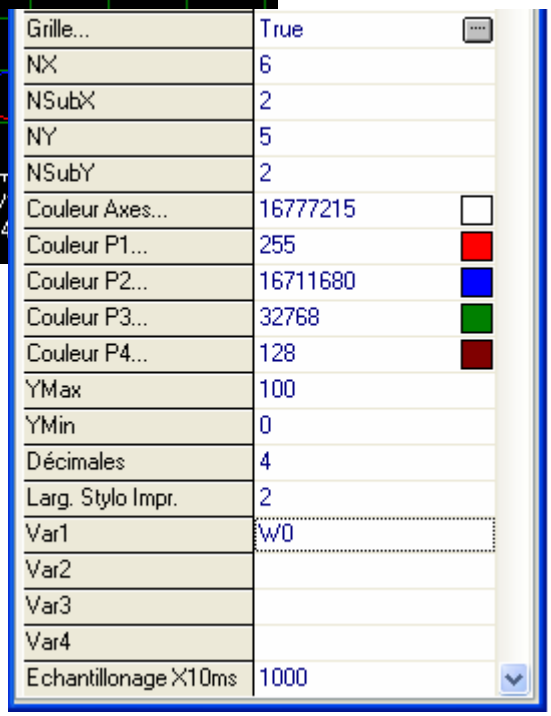
Dans ce cas la valeur renvoyée dans TS est Visual I/O. cependant si la section Software n'existe pas ou le keyname (Name 1) n'existe pas une valeur de chaine par défaut est renvoyé (ici `'ARSOFT Visual'`).

IniFileReadString permet de lire une valeur chaîne à partir d'un fichier INI. Section identifie la section du fichier qui contient la clé voulue. Ident est le nom de la clé dont on doit lire la valeur. Default est la valeur chaîne à renvoyer dans les situations suivantes: ·
Si la section n'existe pas. Si la clé n'existe pas.· Si aucune valeur de donnée n'est affectée à la clé.

Le composant expérimental Trend Histo







Grille affiche ou non la grille. La couleur de la grille est fixée par la couleur de tracé.
Couleur P1, P2, P3, P4 Couleurs des stylos de traçage des courbes.
NX,NSubX,NY,NSubY : Nombre de graduations principales et graduations secondaires sur les axes X et Y.
YMax et Ymin Minimum et maximum de l'axe Y.
Decimales : Nombre maximale de décimales à afficher sur les textes de l'axe Y.
Larg. Stylo Impr. Largeur du stylo en cas d'impression des courbes.
Var1 à Var4 : (optionnel) Nom des variables à afficher au fil de l'eau par le composant, si celui-ci doit être utilisé en traceur.
Echantillonage : Valeur à multiplier par x10ms pour l'échantillonnage et l'affichage de nouvelles valeurs (Var1 à Var4).

Ce composant permet un affichage au fil de l'eau ainsi que la visualisation d'historiques à l'aide de composants additionnels qui seront groupés (clic droit souris menu grouper). Les composants prévus et groupés avec ce composant permettront de créer un visionneur complet et personnalisé.
Un buffer Circulaire pour les 4 courbes (Points X = Date Y=Valeur) de 21600 enregistrements est disponible. Si une seule valeur au fil de l'eau est paramétrée, cette valeur sera bufférisée sur 21600 derniers enregistrements. Si 2 valeurs sont paramétrées alors 21600 / 2 = 10800 enregistrements seront attribués au 2 variables, si les 4 variables (Var1 à Var4) sont paramétrées alors chaque buffer sera de 21600 /4 = 5400 enregistrements.

VPU SHIFTTREND.PAS
Permet l'interface avec ce composant. Les procédures et fonctions de pilotage sont basées sur l'envoi de messages windows au cadre de courbes. Avant de travailler avec ce type de composant il faut d'abord récupérer son Handle par la fonction SHTGetTrendHandle : Integer; pour ajouter des points supplémentaires au cadre de courbe. Pour les autres fonctions il faire être positionné sur le composant pour exécuter celles-ci. Tout est basé sur le passage de paramètres à travers un Large Record.

Exemple d'affichage ou non de la grille :

```
//**** Grid/No Grid **********************
Procedure SHTGridVisible( On : Boolean);
Begin
  POneTrendAttrib:=SHTGetAttributsADDR;          (1)
  If POneTrendAttrib=0 then Exit;
  TrendAttributs(POneTrendAttrib^).Grid:=On;     (2)
  SHTUpdateTrend;                                (3)
End;
```

Pour passer une nouvelle valeur il faut:
(1) Récupérer le handle du composant. Si celui-ci = 0 alors erreur.
(2) Transtyper ce pointeur afin de modifier l'élément du record. Ici on modifie le flag Grid.
(3) Demander au composant de réactualiser son affichage

Détail de l'enregistrement de dialogue avec le composant.

```
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; //*** Axe Y Maximum
  YLow           : Extended; //*** Axe Y Mini
  Decimal        : Integer;  //*** Nombre de décimales affichées sur l'axe Y
  Step           : Integer;  //*** Ecarts (Pixels) entre 2 points
  ColorAxis      : TColor;    //** Couleur des 2 axes
  ColorTextAxis  : TColor;    //** Couleur des textes sur les Axes X et Y
  FontHeightX    : Integer;   //** Hauteur des textes de l'axe X
  NBCourbes      : Integer;  //*** de 1 à 4 maxi 4 aujourd'hui
  ColPens        : Array [1..MaxCourbes] of TColor;   //** Couleur de chaque Axe
  Scales         : Array [1..MaxCourbes] of Extended; //*** Coefficient pour
                        chaque point (original coefficient =1)
  ListPoints     : Array [1..MaxCourbes] of Pointer; //*** Liste des points
  ListPointsHist : Array [1..MaxCourbes] of Pointer; //*** Liste des points
                                                        historiques
  RecordVars     : Array [1..MaxCourbes] of String; //** Noms des variables à
           visualiser au fil de l'eau (Var1 à Var4 dans boite de propriétés)
  FirstPtVisible : Integer; //*** N° du Premier point visible gauche courbe
  BarChannel     : Integer; //*** N+ de Courbe associée à la barre verticale
                             mobile (curseur vertical)
  XChannel       : Integer; //*** Quel courbe Gradue l'axe X
  HintVisible    : Boolean; //*** Bar verticale visible
  XYPlot         : Boolean; //*** Affichage DateTime ou Point sur l'axe X
  inHisto        : Boolean; //*** En historisation ou au fil de l'eau
  SampleTime     : Integer; //*** Echantillonage X10 ms pour affichage fil de
                               l'eau
  StopRecord     : Boolean; //*** Stop ou marche affichage au fil de l'eau
  GotoEnd        : Boolean; //*** Aller et visualiser le dernier point enregistré
  //***** Informations retourné par le système en lecture seule ***************
  RetFrameWidth  : Integer;  //*** Retour de la largeur du cadre de traçage
  RetAdrPosi     : Integer;  //*** Pointer – affichage Position en % sur le fifo
  RetNewPos      : Integer;  //*** Pointer - Position sur le fifo demandé
  RetPosCursor   : Integer;  //*** Position Cursor vertical
  RetPtCursor    : Integer;  //*** Pointeur sur Couple Date/Valeur
  RetPtPoint     : Integer;  //*** Pointer sur le Couple Date/Valeur premier
                              point affiché
  RetMaxPoint    : Integer;  //*** Nombre maximum de points enregistrés par canal
  RetMaxiY       : Extended; //***  Valeur Maxi en Y
  RetMiniY       : Extended; //***  Valeur Mini en Y
End;
```

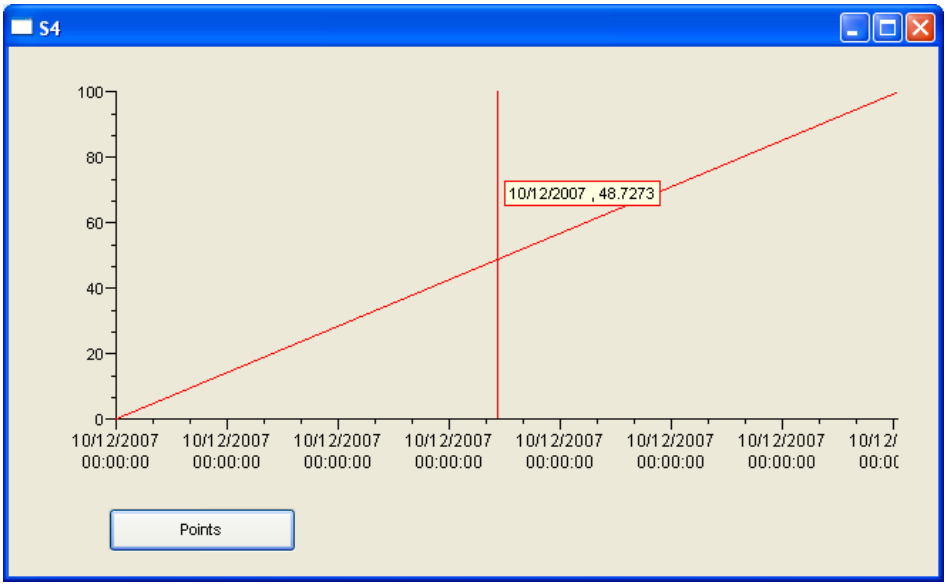
RetAdrPosi : Si <>0 est un pointeur sur une variable Integer qui est rafraîchi automatiquement par le composant. Cette variable indique la position (en %) du premier point affiché dans le fifo (le plus à gauche dans le composant).
RetNewPos : Si <>0 est un pointeur surveillé par le composant lui indiquant en cas de changement de celui-ci la nouvelle position (en %) dans le fifo courant pour démarrer son affichage de la gauche vers la droite.

Pour tout ajout de fonctionnalités se reporter au source fourni SHIFTTREND.PAS. Ce composant est susceptible d'évoluer avec les futures versions de Visual I/O.

Quelques composants en créations.
Disponibilité Mars 2007
Onglet Trends

Ajouter à une courbe des points par programme.

Cet exemple permet de tracer par programme des points (courbe linéaire) dans un visualisateur de courbe sans utiliser les composants de l'onglet trends. Une détection autoamtique de la largeur de la zone utile du tracé permet d'ajouter le nombre de points en conséquence. Placez simplement une courbe sur le synoptique.



Puis le programme derrière le bouton

```
{** 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;
```

Les Chaînes longues

Visual I/O utilise de base le type String, qui est une chaîne de caractères courte (255 caractères maximum) avec comme premier byte [0] la longueur de la chaîne. Il est cependant dans certains cas nécessaire de disposer de chaînes de caractères de grandes longueurs (>255 car). Le Type PChar pointeur sur chaîne de caractères est possible tout simplement en utilisant la procédure Getmem qui alloue un nombre d'octets dans la mémoire. La difficulté est alors de manipuler ces PChar comme des chaînes de caractères normales.

Beaucoup de langages de programmation, dont le C et le C++, n'ont pas de type de donnée chaîne naturelle. Ces langages, et des environnements comme Windows qui ont été construits avec, utilisent des chaînes à zéro terminal. Une chaîne à zéro terminal est un tableau de caractères d'indice de base zéro et terminé par NULL (#0). Comme le tableau n'a pas d'indicateur de longueur, le premier caractère NULL indique la fin de la chaîne. Vous pouvez utiliser les routines spéciales de l'unité Sysplc pour manipuler des chaînes à zéro terminal quand vous avez besoin de partager des données avec des systèmes les utilisant. La taille maximum d'une chaine de caractères à zéro terminal est de 2Go maximum.

Quand on parle de PChar ou de chaines à zéro terminal ou de pointeur sur chaine de caractères onparle de la même chose.

La function StrToPchar transforme une chaine de caractères classique en pointeur sur chaine (Pchar).
Exemple :

```
Var
  P : Integer;
Begin
  P:=StrToPChar('Visual I/O');
```

La function StrAdd ajoute une chaîne de caractères à un Pchar.

```
P:=StrAdd(P,' Development Tool');  //**P Pointe sur 'VisualI/O Development Tool'
```

StrLen : Renvoi la longueur de la chaîne Pchar.
StrEnd : Renvoi l'adresse du dernier caractère dans la chaine PChar.
StrComp Compare les 2 Pchar et retour ne 0 si égal.
Le retour <0 Si Str1< Str2, =0 si Str1= Str2, >0 si Str1 > Str2
StrMove Copie exactement Count caractères de Source vers Dest et renvoie Dest. Source et Dest peuvent se recouvrir
StrAlloc Alloue un tampon pour une chaîne terminée par le caractère Null et renvoie un pointeur sur le premier caractère de la chaîne.
StrBufSize Renvoie le nombre de caractères pouvant être placé dans un tampon alloué par StrAlloc
StrNew Alloue une copie de Str dans la mémoire.
StrDispose Restitue une chaîne précédemment allouée à l'aide de StrAlloc ou StrNew
StrConCat Concat Destination + source et renvoi le pointeur sur le premier caractère.
StrCopy Copie Source vers Dest et renvoie Dest

Liste des routines de gestion de chaînes (terminées par le caractère Null)

```
//****** 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;
```

La librairie WinSock

La librairie winsock.pas permet une interface simplifiée et efficace avec les sockets de windows. Un socket grâce aux procédures et fonctions de winsock.pas s'apparente énormément à une liaison série. Avec une ouverture de port, des lectures ou des écritures suivie d'une fermeture de port. Attention cette librairie est utilisée par OpenModbus. Si vous désirez modifier les fonctions créez une nouvelle librairie (par exemple Winsock2.pas).

Les procédures et fonctions

la combinaison **adresse IP + port** est alors une adresse unique au monde, elle est appelée **Socket**

```
//*** Ouvre un socket sous windows.
Function SocketOpenConnection (IPA : String; Port : Integer; Var DSocket :
Integer) : Boolean;
IPA    : Adresse IP;  Port: Port de communication; Retour: True si socket ouvert.
//*** Ferme le socket.
Procedure SocketCloseConnection ( Var DSocket : Integer);

//*** Attends un nombre d'octêts précis venant du socket ou 500ms de time out.
Function SocketWait (Var SocketHandle : Integer; NB : Integer) : Boolean;

//*** Lit le buffer de réception du socket.
Function SocketReceiveBuf (Var SocketHandle : Integer; var Buf; Count: Integer):
Integer;
//*** Envoie des octets dans un socket
Function SocketSendBuf(Var SocketHandle : Integer; var Buf; Count: Integer):
Integer;
//*** Retourne le nombre d'octêts dans le buffer de réception du socket
Function SocketReceiveLength (Var SocketHandle : Integer) : Integer;
//*** Retourne l'adresse IP du PC
Function GetHost_IP     : String;  //*** Exemple "192.168.0.1"
//*** Retourne le nom du PC
Function GetHost_Name     : String;  //*** Exemple "ShoeBox"
```

Exemple écriture et lecture dans un 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); //** Raz buffer local d'émission
     Obuf[5] :=3;
     Obuf[6] :=12;
     Obuf[7] :=7;
     Obuf[8] :=70;
     I:=SocketSendBuf(DSocket,OBuf,9); //** envoi de 9 bytes
     If I<>-1 Then     //** si envoie est ok alors I<>-1
     Begin
      If SocketWait(DSocket,8) Then //** Attente de 8 Bytes minimum en retour
      Begin
        I:=SocketReceiveBuf(DSocket,IBuf, 500); //** Récupère le buffer entier
                                                // maximum 500 bytes
        If I>=8 then //I contient le nombre de bytes lus
        Begin
          Move(Ibuf[0],TabloRetour[0],I); //** exploite le buffer récupé
        End;
     End;
     SocketCloseConnection(MySocket); //** Ferme le socket
  End;
End;
```

Affichage sur plusieurs moniteurs.

Visual I/O permet la gestion de plusieurs moniteurs raccordés à votre PC.
Vous pouvez alors dispatcher vos synoptiques sur ces différents moniteurs.
Quand un synoptique est affiché sur un des moniteurs, les autres synoptiques lancés à partir de ce même synoptique seront eux aussi affichés sur ce moniteur. Cela permet de gérer la navigation sur le même écran (Moniteur).

Considérons un système à 3 moniteurs, vous devez afficher un premier synoptique qui est similaire sur chaque moniteur. Vous allez développer le synoptique une seule fois et vous utiliserez la procédure de duplication DuplicateForm.

Exemple :

```
If Button Then
Begin
  Display('MainMenu'); //** Affichage sur écran principal
  Duplicateform ('MainMenu',2,2);       //** Le même sur Ecran N°2
  Duplicateform ('MainMenu',3,3,20,550); //** Le même sur Ecran N°3 X=20 & Y=550
End;
```

Procedure DuplicateForm (**NomSyno** : String; **Indice** : Integer; **Moniteur** : Integer; **NewPosX, NewPosY** : Integer);

Nomsyno : Nom du synoptique à dupliquer.
Indice : Numéro d'indice ajouté à la fin du nom du synoptique dupliqué.
Moniteur : Numéro du moniteur. Le moniteur principal porte le Numéro 1.
NewPosX : Nouvelle coordonnée en X sur le monitor (Optionnel).
NewPosY : Nouvelle coordonnée en Y sur le monitor (Optionnel).

Un nouveau synoptique est créé dans l'application, celui-ci est rigoureusement identique à l'original sous le nom NomSyno+Indice.

Exemple `Duplicateform ('MainMenu',2,2);` Le nom de synoptique qui sera affiché sur le moniteur numéro 2 est MainMenu2.

Dans l'exemple ci-dessus comme MainMenu est dupliqué en MainMenu2 et MainMenu3 il peuvent appeler individuellement les mêmes synoptiques. Si un Mainmenu affiche un synoptique, celui-ci s'affiche sur le même moniteur. Si un mainmenu d'un autre moniteur appelle le même synoptique, celui-ci est alors déplacé du moniteur sur lequel il est affiché sur le nouveau moniteur. En d'autres termes le synoptique ne peut pas être affiché sur 2 moniteurs en même temps.

Si nécessaire il faudra aussi dupliquer les synoptiques à afficher en multi instances.

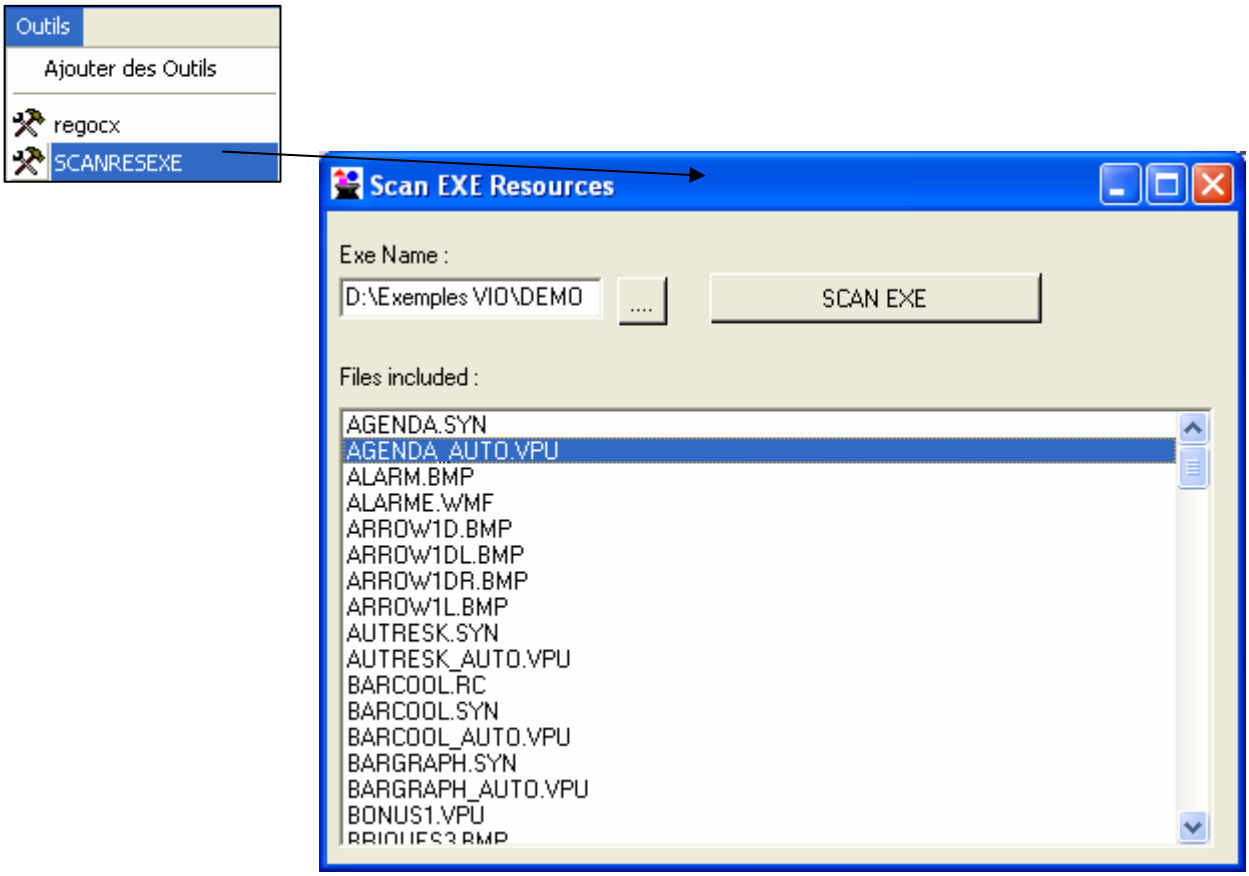
Procedure **DisplayEX**(NomSyno : String; Moniteur : Integer);
Cette procédure est une extension de la procédure Display. Elle permet d'afficher un synoptique sur un moniteur spécifique. Si le moniteur n'existe pas, le synoptique sera affiché sur le moniteur principal.
Le paramètre peut être absent, dans ce cas cette procédure à la même fonctionnalité que celle de Display.

Exemple:

```
DisplayEX ('SumAlarms',2);  //** Affiche le syno SumAlarms sur le moniteur N° 2
DisplayEX ('Trends'); //*** Affiche le syno Trends sur le moniteur principal.
```

Utilitaire SCANRESEXE.EXE

Ce programme permet de scruter la ressource de l'exécutable créée avec Visual I/O.
Cet utilitaire permet ainsi de contrôler si tous les fichiers spécifiés sont bien inclus dans l'exécutable final qui sera transporté sur le PC final de l'application.
Si cet utilitaire n'est pas présent dans votre PC, téléchargez le sur le site ARSOFT par la commande Mise à jour



Choisissez l'exécutable en cliquant sur le bouton à droite du champ de saisie, puis cliquez sur SCAN EXE.
La listbox inférieure affiche tous les fichiers inclus en ressource dans l'exécutable créé par visual I/O.

☞ **Note** : Cet utilitaire permet seulement la visualisation.