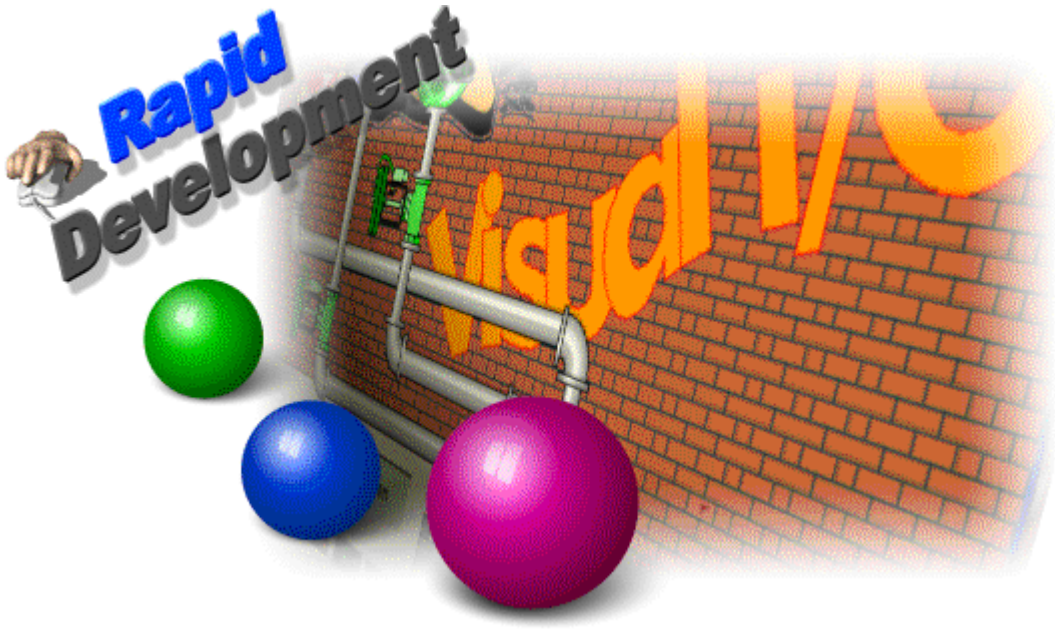
Visual PLC

Productivity tools

Manuel de Programmation



- **Editeur**
- **Language**
- **Jeu d'Instructions**

Généralités

Visual PLC est un outil d'automatisme alliant simplicité, puissance et rapidité de traitement.
Arsoft ayant acquis une longue expérience de l'automatisme par PC sous l'environnement MS-DOS, à fait migrer ses développements sous Windows 95,98, NT, 2000, XP et XPE.
Pour satisfaire au besoin de tous type de traitements ainsi qu'aux impératifs fonctionnels de l'automatisme, Arsoft à développé son propre compilateur optimisé pour ceux-ci. Grâce à son compilateur Visual PLC est un outil unique.
Arsoft maîtrisant le développement sous des OS complexes comme Windows NT, XP, à développé son propre noyau temps réel préemptif et sécurisé. Avec sa gamme et Visual I/O, Arsoft offre une suite logicielle cohérente dans le domaine de l'automatisme des machines de production à fortes cadences et très souvent temps réel.

Visual PLC est un système 32 Bits, c'est à dire qu'il ne fonctionne que sous les systèmes d'exploitation Windows 98 et NT,2000 et XP (Windows 3.X étant exclu).

Windows XP est fortement recommandé pour faire fonctionner Visual PLC.

Structure des programmes et des données

Visual PLC incluant le compilateur Arsoft, fonctionne d'une façon différente des programmes compilés classiques.

Tous les programmes que vous allez écrire, seront compilés au vol. C'est à dire qu'il n'y aura aucun temps d'attente ou de programmes lancés en externe.
Chaque programme (Ladder, Grafcet, Langage etc.) sera vérifié en syntaxe puis compilé dans un fichier d'extension. **VPU.**
L'application finale peut être aussi constituée par un seul fichier .exe

Les fichiers VPU (Visual Pascal Unit).

Ces fichiers sont les résultats de compilation de programmes. Ces fichiers compilés contiennent toutes les déclarations de variables et de procédures exploitables par le noyau temps réel ou par d'autres VPU. Par exemple une VPU dans sa forme la plus simple est la déclaration de variables globales de l'application. Cela peut concerner aussi des procédures ou des fonctions que vous avez créées pour l'exploitation dans d'autres programmes. Les VPU étant des fichiers pré-compilés, les programmes les utilisant n'ont pas besoin de disposer de leurs codes sources. Il est impossible de savoir sous quel langage une VPU à été écrite. L'on verra plus tard, que les VPUS ont une partie Interface et une partie Implémentation.
L'interface est la déclaration des variables ou des procédures accessibles par d'autres VPU. L'implémentation est la partie réelle du code.
Les programmes 'Drivers' communiquant avec les cartes embarquées dans le PC comme la carte Profibus sont aussi des VPUS permettant de faire l'interface avec des éléments physiques (E/S) et la table image des entrées/Sorties qu'utilise les programmes.

MVARGLOB.VPU

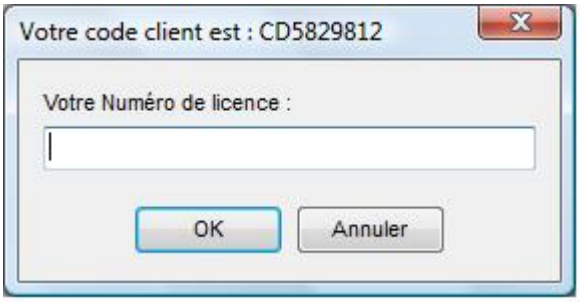
Au sein de Visual PLC (et Visual I/O), MVarglob.VPU est le fichier de base contenant toutes les variables du Process. Tel que la table image des entrées sorties, les variables internes, les bits des étapes actives et leurs temps d'activation. Avant de commencer à développer un programme en Ladder, Grafcet ou autre, il est nécessaire de déclarer des variables à l'aide de la rubrique Variables Globales.
Des modèles de variables (pré-définitions) ont été implémentés. Au démarrage d'un nouveau projet, vous pouvez choisir un modèle (Télémécanique, Siemens, Allen Bradley Etc..).
A ce modèle vous pouvez aussi rajouter vos propres variables.

Licence runtime de Visual PLC

L'éditeur de Visual Plc n'est pas soumis à licence seul le moteur d'exécution est soumis.
Si Visual PLC est utilisé avec une carte ARSOFT le Runtime ne nécessite pas de licence car elle est intégrée dans la carte ARSOFT.

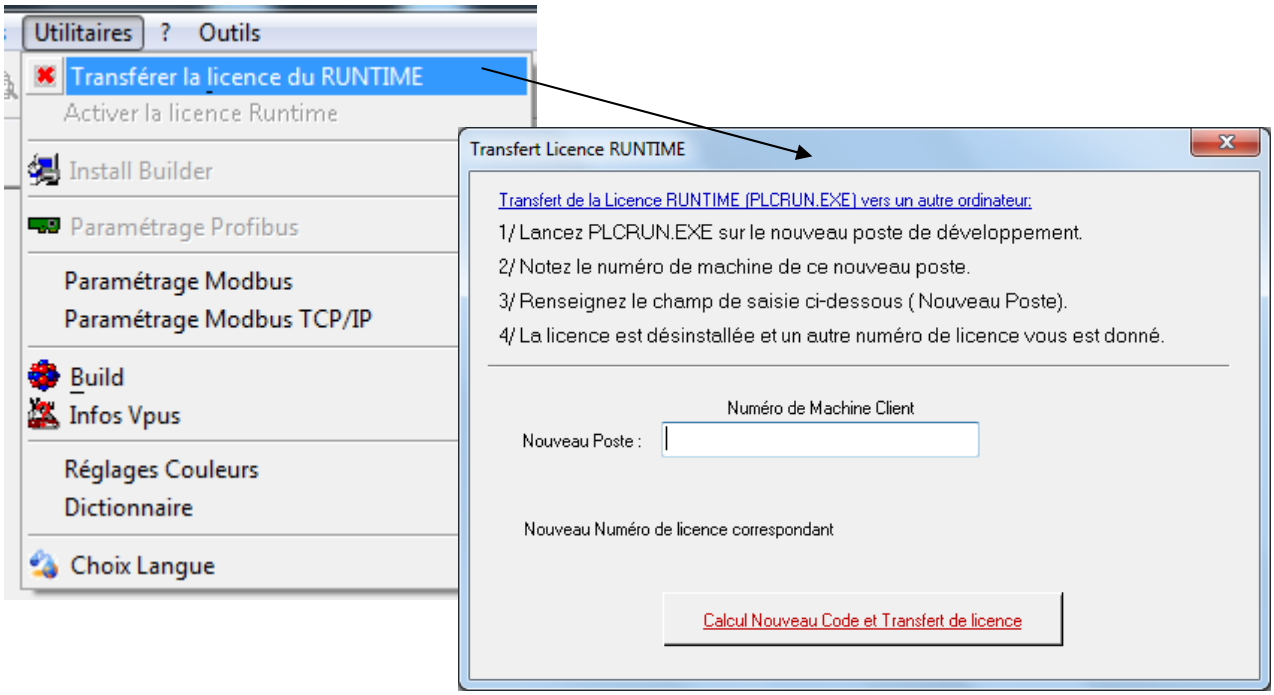
Cas ou licence dans un PC.
Vous avez besoin d'un code d'activation.

Menu Utilitaires → Activer la licence RUNTIME



Notez le code client généré sur ce PC (dans l'exemple CD5829812) transmettez le à la hotline ARSOFT hotline@arsoft-int.com afin que celle-ci vous transmette en retour le code d'activation.

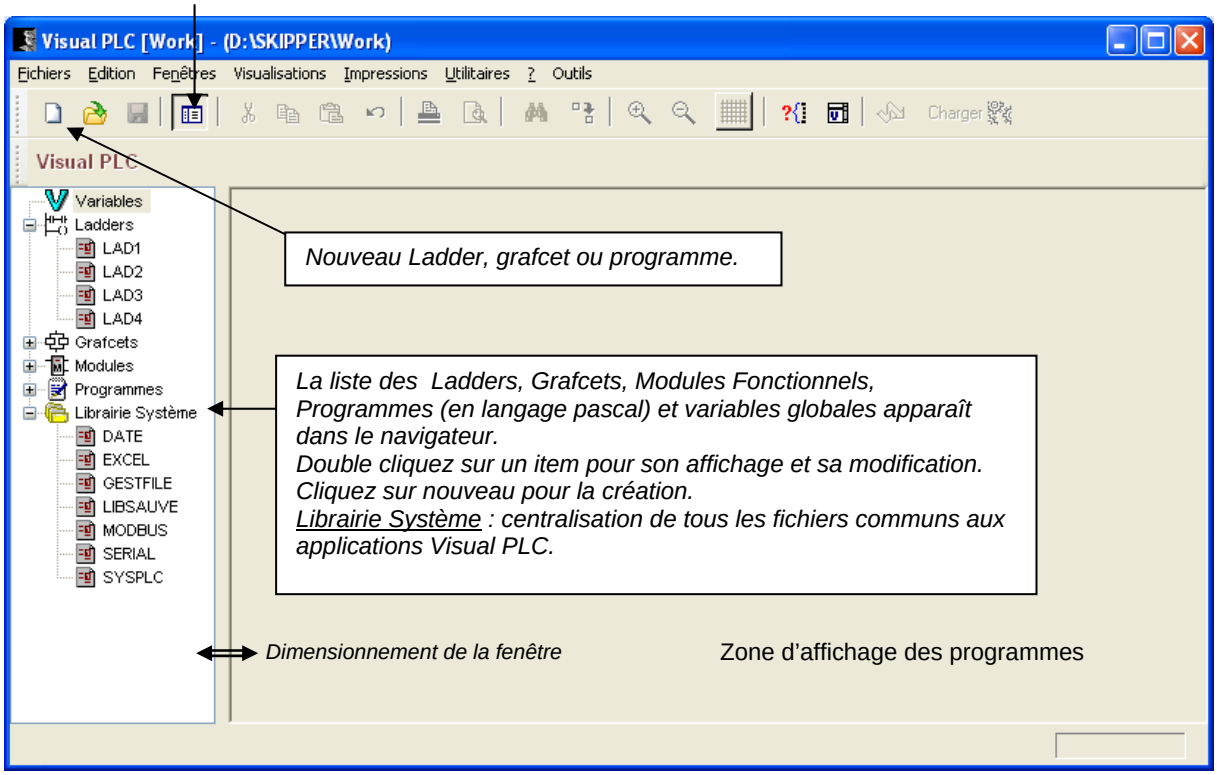
Dans le cas ou vous changerez de PC, cliquez dans le menu transférer la licence du RUNTIME

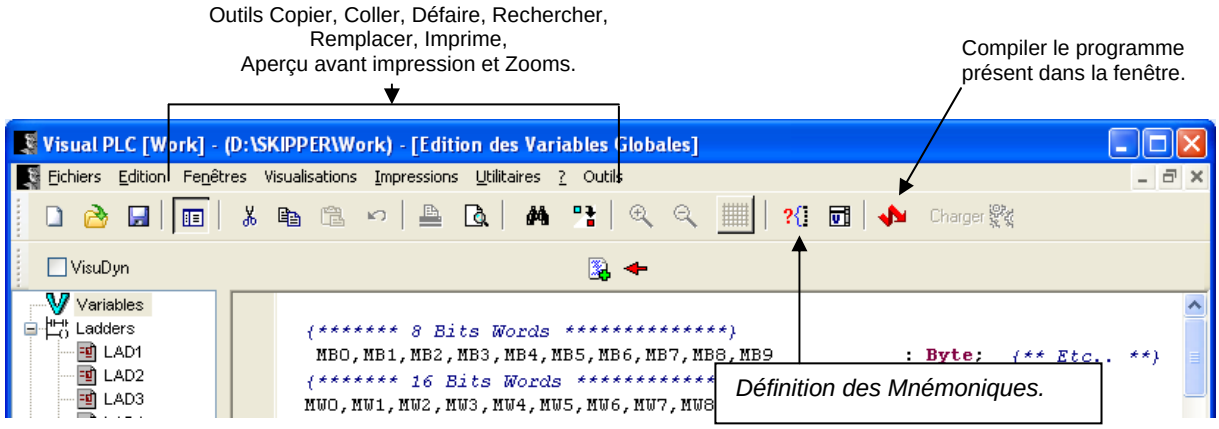


Le nouveau PC doit être installé au préalable afin de récupérer le code client final. Frappez ce code dans le champ de saisie Nouveau Poste. Un nouveau code d'activation est généré et le PC d'origine est désinstallé.

Présentation de l'éditeur

Affichage du navigateur



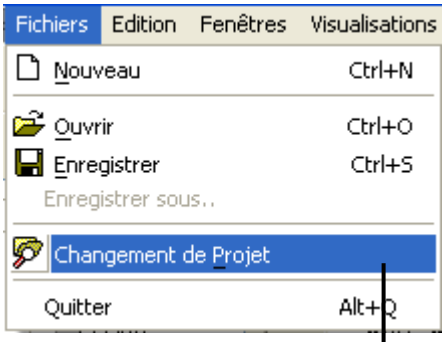


Première Etape

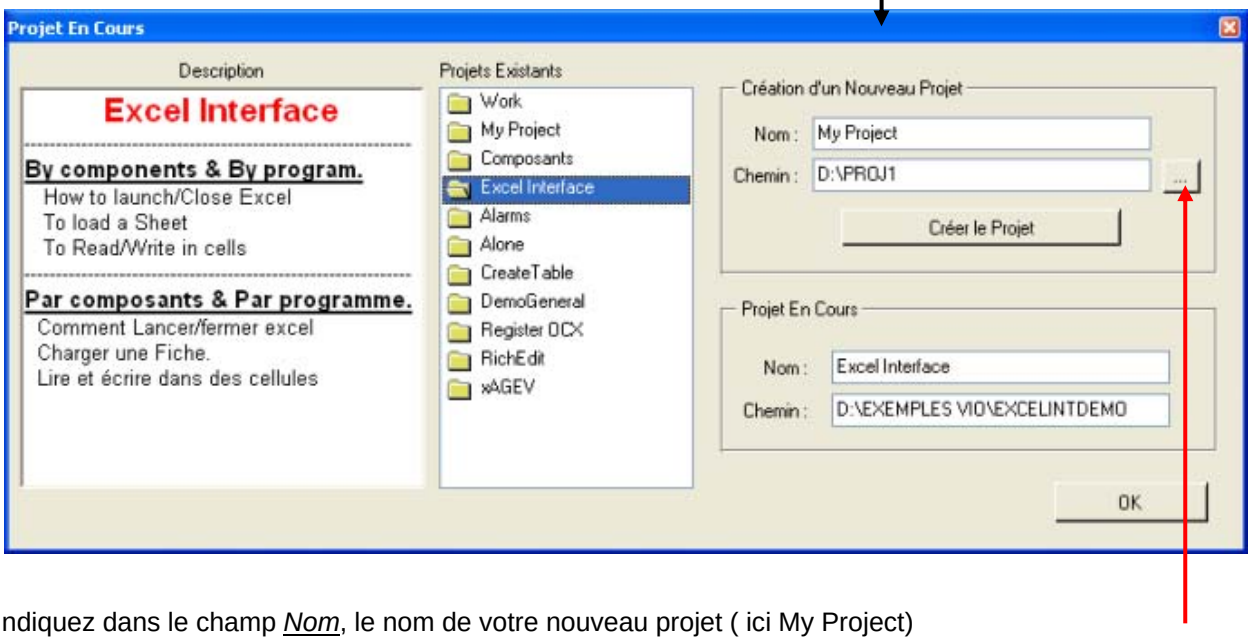
Définir le nom du projet

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



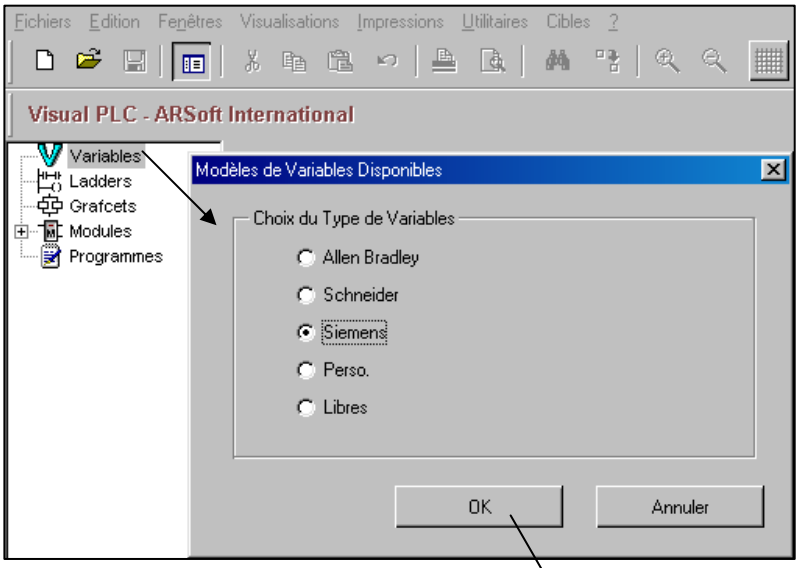
Indiquez dans le champ *Nom*, le nom de votre nouveau projet (ici My Project)
Indiquez dans le champ *Chemin*, le chemin complet ou doit se situer ce nouveau projet (ici D:\Proj1) par navigation
En cliquant sur le petit bouton. Si le répertoire n'existe pas vous avez la possibilité de le créer vous avez 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

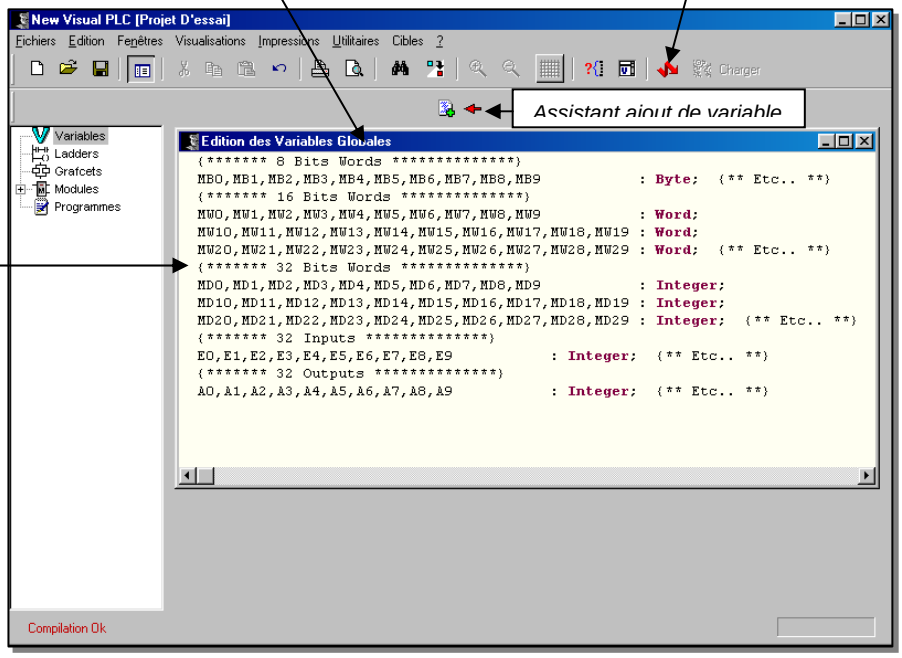
Deuxième Etape

Déclaration des variables du process (Variables Globales).



Double-Cliquez sur Variables. Une fenêtre apparaît vous proposant un modèle de variables pré-définis.

Compilation des Variables. (Crée Mvarglob.VPU)



Le système crée une liste de variables correspondant à l'automate spécifié. L'utilisateur peut à tout moment supprimer ou rajouter une variable de son choix et écrivant la variable à la suite ou en se servant de l'assistant.

Note : Tous les Programmes Ladder, et Grafcets font référence aux variables globales.

Une variable peut être initialisée.

Exemple : Consigne : Integer=1000 ;

Signification : Au lancement de l'application une variable consigne (32 bits) est disponible et contient la valeur 1000.

Si aucune initialisation n'est spécifiée, le système positionne toutes les Variables Globales à 0.

Type de Variables dans Visual PLC

Dans Visual PLC, 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 PLC 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;
```

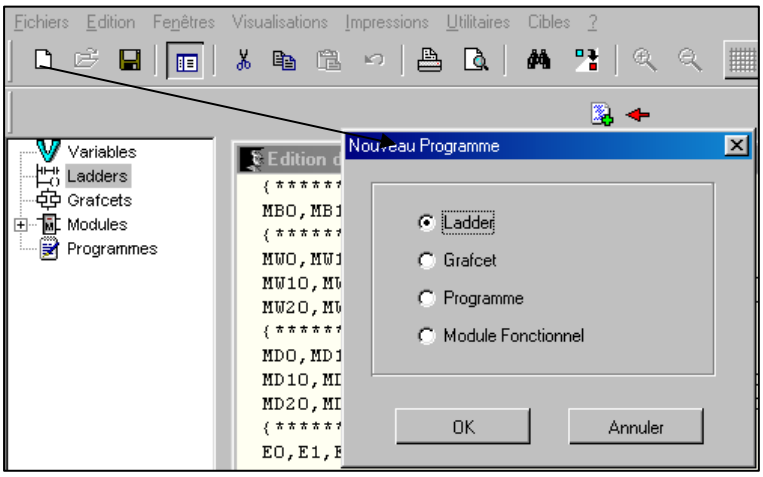
Contrairement à d'autres Soflogic, Visual PLC permet l'adressage Bit dans un mot.

Exemple : W10.0 (premier bit du mot W10) .

Troisième Etape

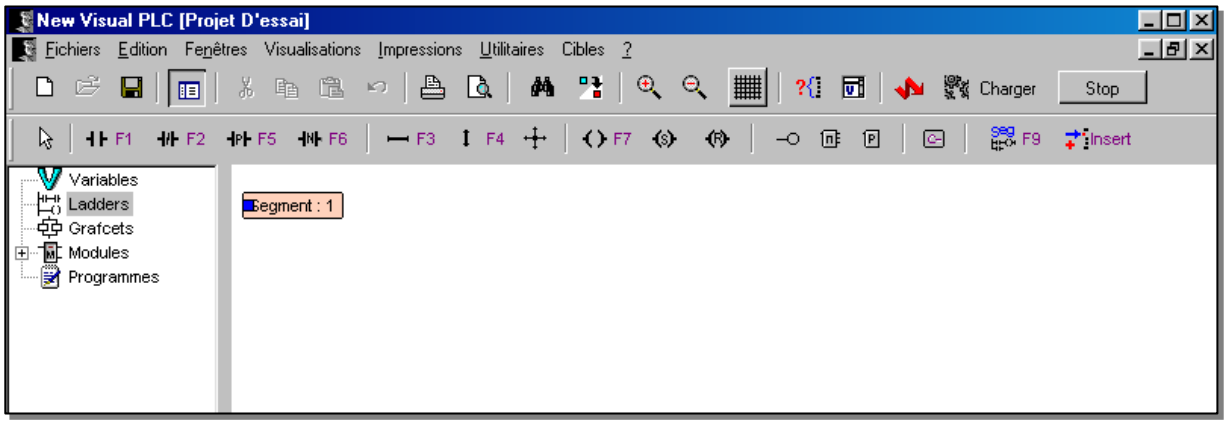
Ecriture d'un programme en Ladder

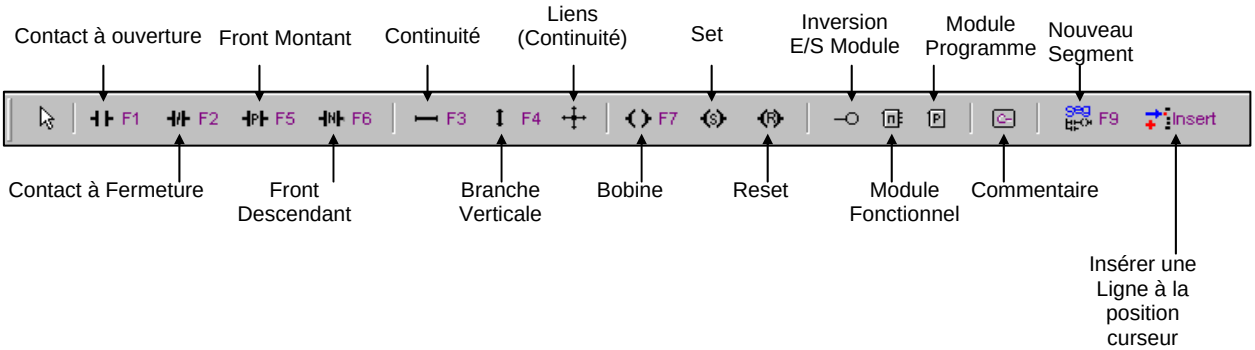
Après avoir défini des variables globales et des variables internes dans MVarglob.VPU, il est possible d'écrire des programmes en Ladder (qui seront eux aussi des VPUs).



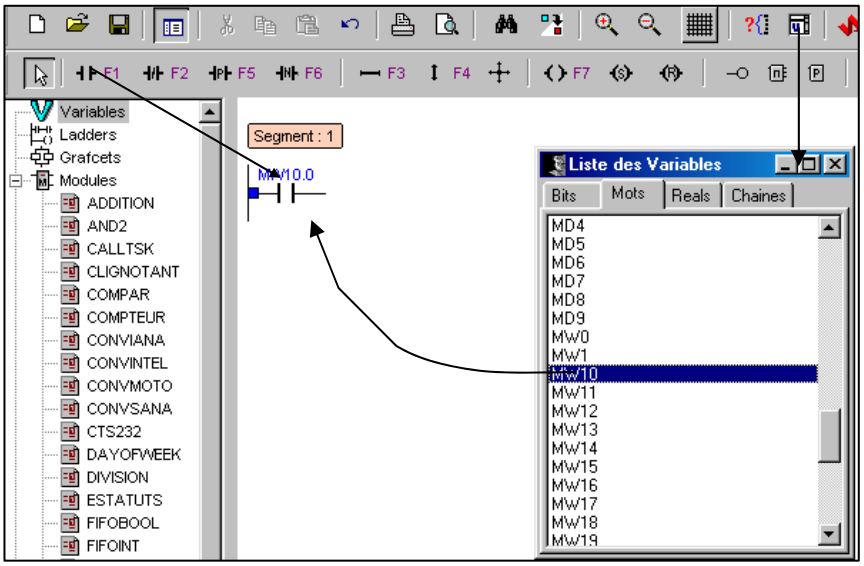
Positionnez vous sur Ladders, puis cliquez sur le bouton Nouveau. Une fenêtre apparaît. Cliquez sur OK.

Une fenêtre permettant d'écrire un programme en Ladder apparaît. Une barre d'outils se positionne en dessous de la barre standard. Les programmes écrits en Ladder, sont organisés en segments consécutifs, permettant ainsi la constitution de références croisées.





Programme Clignotant

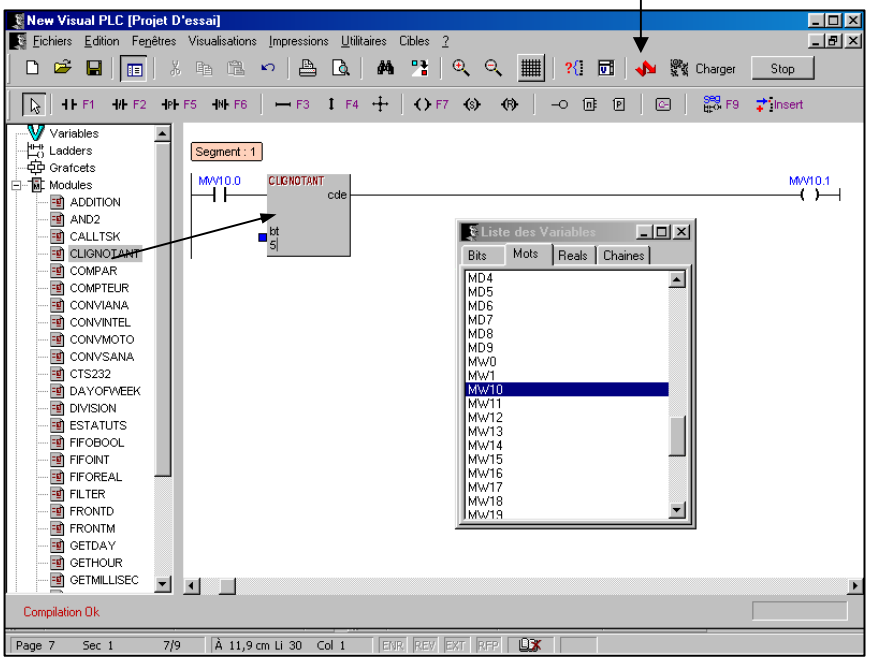


Amenez la liste des variables pour un glisser/déplacer sur le contact. La frappe du contact est aussi possible au-dessus du contact.

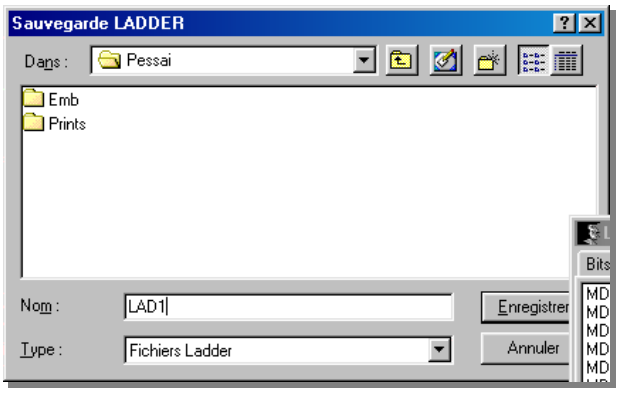
Compilation du Ladder Visible.

Amenez par Glisser/Déplacer le module fonctionnel 'Clignotant'. Placez une bobine en sortie en utilisant le bit 1 du mot MW10.

Réglez la valeur de la base de temps du module (en 1/10 sec.). Ici 5 (0.5 Secondes). En final Compilez votre Ladder en cliquant sur le bouton Compiler.

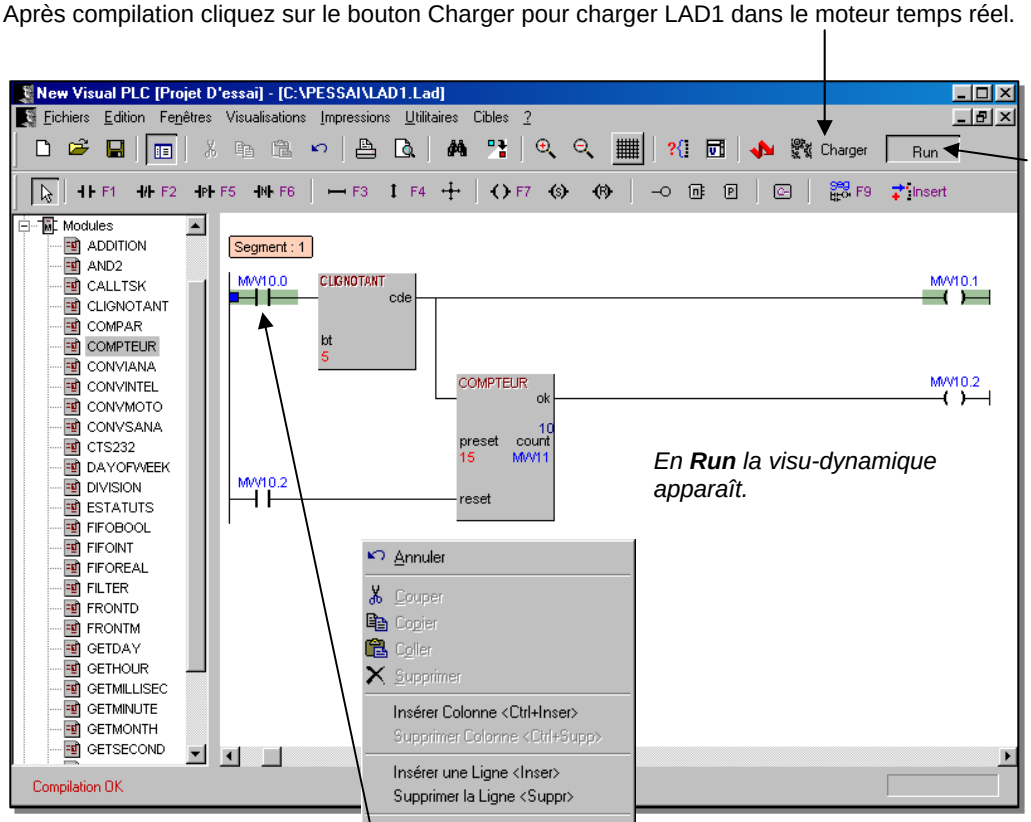


Lors de la première compilation, le système vous demande un nom pour celui-ci. Une boîte de dialogue apparaît comme ci-contre.
*A chaque Compilation le système sauve le programme puis le compile en créant un fichier compilé d'extension .**VPU**.*



Chargement dans le moteur temps réel.

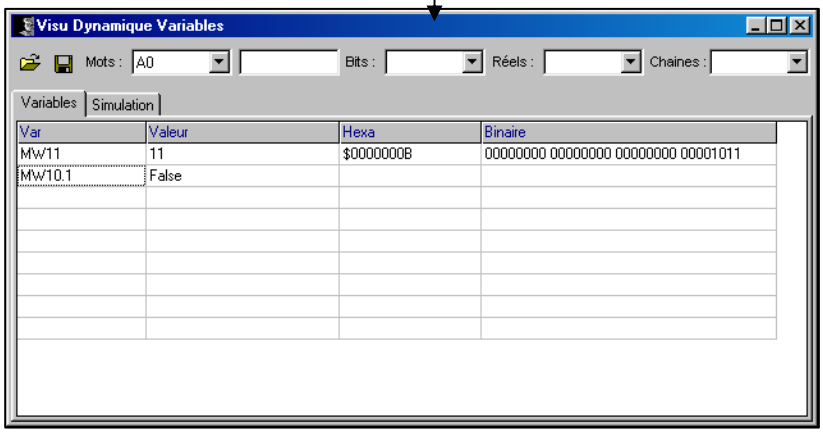
Après compilation cliquez sur le bouton Charger pour charger LAD1 dans le moteur temps réel.



Run/Stop Lad1 dans le moteur temps réel.

*En **Run** la visu-dynamique apparaît.*

Par un clic droit à la souris, un menu flottant apparaît permettant d'inverser l'état d'un bit pointé par le curseur. Ici MW10.0 est forcé à 1 (true).



Une grille de visualisation/Forçage est aussi disponible en sélectionnant Visu Variables dans le menu flottant. Ces commandes sont disponibles aussi dans le menu général de Visual PLC.

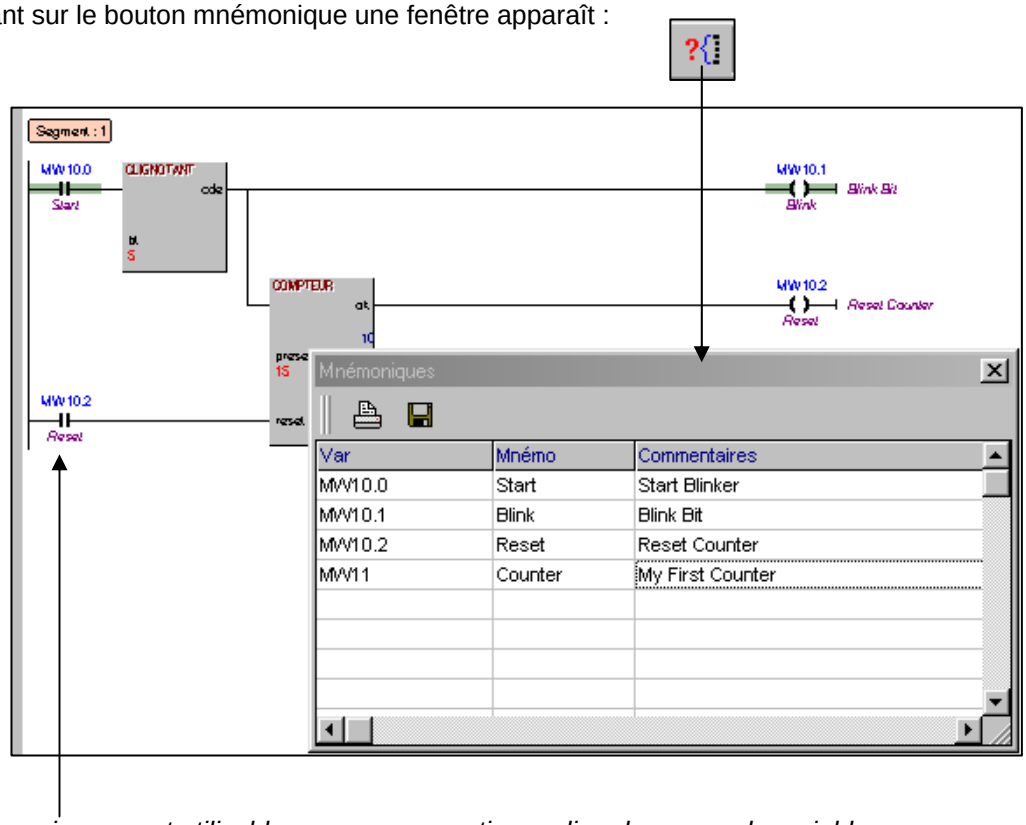
Note : Sauvegardez les listes de variables visualisées a l'aide du bouton Sauver.

Autre :

Dans notre exemple 2 modules fonctionnels sont en cascades.
Visual PLC permet la cascade de modules sans affectation de variables intermédiaires.
Les modules fonctionnels nécessitent des variables en paramètres (Variables ou constantes). Si le programmeur décide de ne pas affecter de variables, le compilateur crée automatiquement des variables fantôme permettant au module de fonctionner normalement. Ces variables fantôme seront visibles en visu dynamique mais pas exploitables par programmes.

Mnémoniques

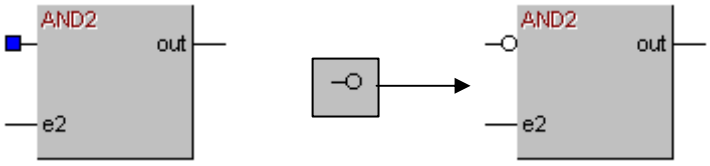
MW10.0,MW10.1 et MW10.2 ne sont pas des noms très significatifs.
Visual PLC permet l'affectation de mnémoniques à ces variables ou à ces bits de variables.
En cliquant sur le bouton mnémonique une fenêtre apparaît :



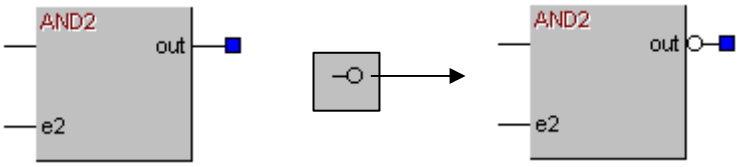
Ces mnémoniques sont utilisables en programmation au lieu des noms de variables.

Inversion d'entrée ou de sortie d'un module fonctionnel

Cette commande permet d'inverser l'entrée ou la sortie d'un paramètre bit d'un module fonctionnel.
1/ placez-vous sur le paramètre bit choisi d'un module fonctionnel puis cliquez sur ce bouton



Avant **Après**



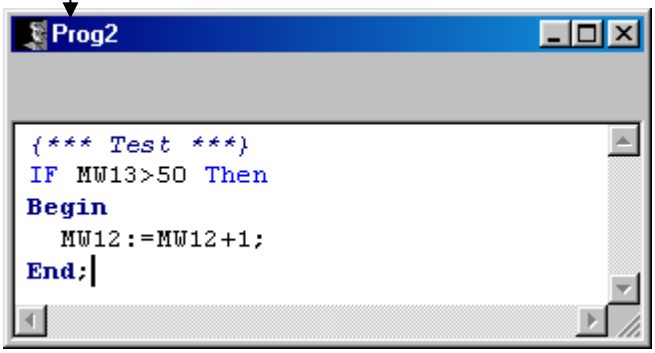
Exemple : Transformation d'un Bloc ET en Bloc NAND.

Blocs Programmes

Il est possible d'insérer des blocs programmes dans des lignes d'un Ladder.

Cliquez sur le bouton Bloc Programme puis placez ce bloc (vide) dans le Ladder courant. 

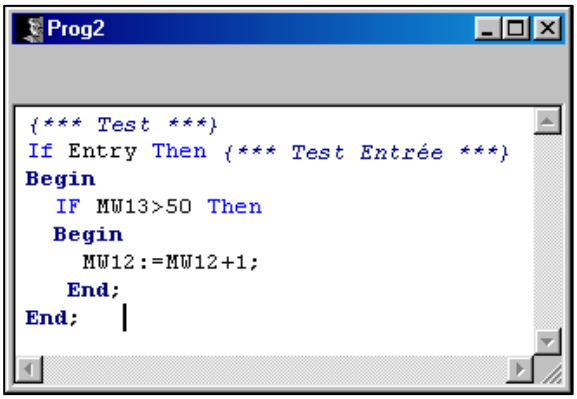
Double-cliquez sur ce bloc, un éditeur de programme apparaît :



Ce bloc programme est toujours exécuté même si l'entrée est à False (0).

Le résultat de l'entrée est transmis à la sortie

Possibilité de tester si l'entrée du bloc programme est Vrai.



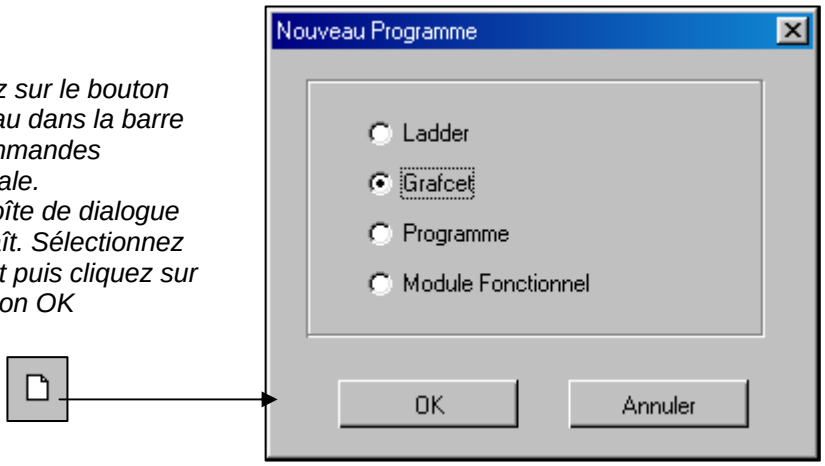
*Le mot réservé **ENTRY** (Boolean) permet de tester l'entrée du bloc programme et réagir en conséquence. Ici le test MW13>50 est exécuté si l'entrée du bloc programme est vrai.*

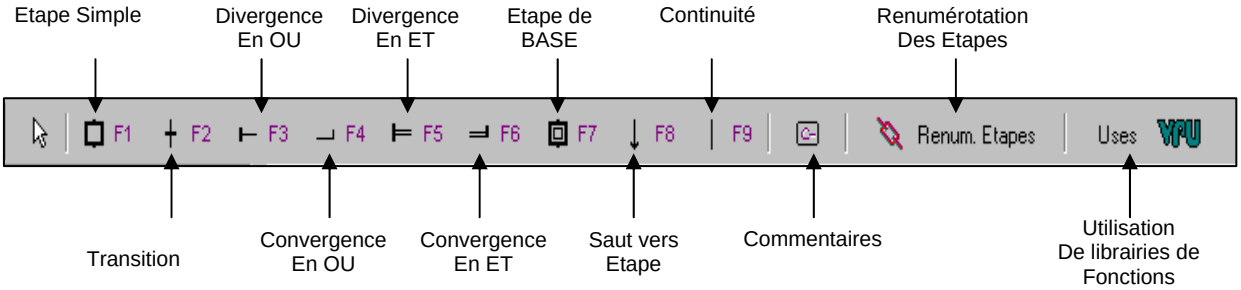
Quatrième Etape

Ecriture d'un programme en Grafcet

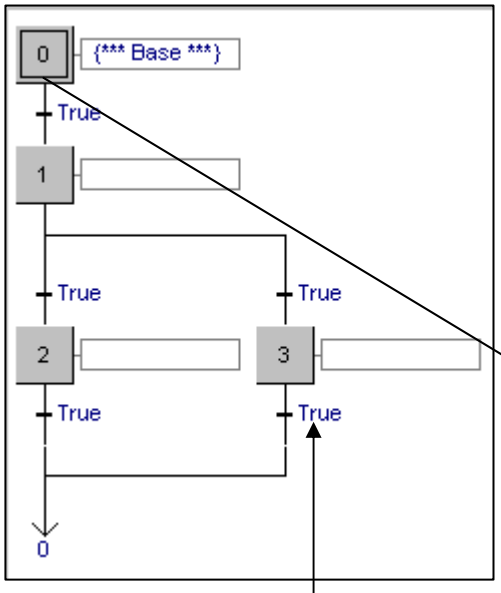
Après avoir défini des variables globales et des variables internes dans MVarglob.VPU, il est possible d'écrire des programmes en Grafcet (qui seront eux aussi des VPUs).

Cliquez sur le bouton nouveau dans la barre de commandes principale.
Une boîte de dialogue apparaît. Sélectionnez Grafcet puis cliquez sur le bouton OK

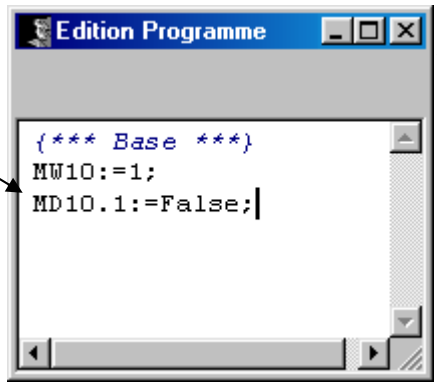




Création d'un GRAFCET



Choisissez l'élément de la barre de commandes à positionner sur l'écran puis placez le à l'endroit voulu. C'est la partie design.
Note : Les étapes de base ne peuvent êtres positionnés que sur la première ligne.



Double-cliquez sur l'élément choisi. Soit une étape ou une transition. Un éditeur de code apparaît. Procédez de la même manière sur tous les éléments du Grafcet.

Si les **transitions** ne contiennent pas de code, le système les rend **passantes**.

Note Importante :

Les Divergences d'un Grafcet ne peuvent partir que vers la droite de l'étape de base et revenir alignées en dessous de l'étape de base.
Un Grafcet ne peut être centré sur axe de symétrie vertical.

Visual PLC permet la programmation en Grafcet. Il respecte les normes Grafcet établies, mais peut traiter tous cas particuliers qui restent à la responsabilité du programmeur.
Le Grafcet contient des équations de transitions, celles-ci étant programmées en littéral à l'aide des instructions AND OR NOT et toutes fonctions évoluées venant des librairies VPU.

On peut avoir une simple équation du type : I0.0 And I0.1
A l'équation complexe du type : I0.0 and **MaFonction**(W0,12,Tab[1])

Mafonction étant une fonction localisée dans une VPU (Librairie).

Visual PLC pourra s'appuyer sur le Grafcet en tant qu'organisateur de cycle ou l'on trouvera des équations complexes en schéma à contacts validant les transitions. Visual PLC pourra dans ses étapes recevoir un programme complet utilisant les mêmes instructions que celles accessibles par les librairies c'est à dire le jeu d'instructions complet du langage Pascal. Les programmes écrits en Grafcet sont compilés comme tout autre programme et stockés au format VPU.
Le moteur temps réel chargeant des VPU ne peut savoir si ce programme à été écrit en Grafcet ou Ladder ou autre langage.

Renumérotation des étapes

L'éditeur fait apparaître les étapes ainsi que les numéros correspondants.
Les numéros d'étapes sont fixés par le système, elles sont modifiables par **renumérotation** automatique (clic sur bouton Renum Etapes).

Attention cette fonction est sous la responsabilité du programmeur.
En effet comme à chaque étape un booléen (bit) reflète l'état de celle-ci, ce bit peut bien sur être exploité dans d'autres programmes.
La renumérotation ne recherche pas ces bits et leurs exploitations **dans les autres programmes.**

Instructions et variables concernant un Grafcet

Lorsque vous programmez en Grafcet, vous avez la possibilité de récupérer ou de forcer certaines informations le concernant.

Détection d'appel d'une étape

La variable interne START_**STEP** passe à True lorsque l'étape est appelée.
Ce programme placé dans une étape permet de détecter la validation d'une étape quelconque.

Ex: If **Start_Step** Then Compteur:=0;
{Au front d'appel de l'étape le compteur est mis à zéro.}

Temps d'activation d'une étape Grafcet.

Le temps d'activation d'une étape est contenu dans :

Une variable de type tableau d'integer **TStep**[X]. Ce tableau n'est accessible que dans le Grafcet lui-même.

Ex : If Tstep[1]>=10 Then Alerte:=True;
{ si le temps d'activation de l'étape 1 est >= à 1seconde alors le bit Alerte passe à True}

Variables d'un Grafcet utilisables par les autres programmes.

Il est possible de tester ou de forcer les étapes d'un Grafcet. Visual PLC créée automatiquement un tableau de Booléens et d'integers correspondant aux états des étapes du Grafcet ainsi qu'aux temps d'activations de celles-ci.

Supposons que votre Grafcet s'appelle MonGraf, et qu'il dispose de 10 étapes au total. Ces étapes sont numérotées de 0 à 9.
Visual PLC créée automatiquement deux tableaux désignés dans Mvarglob par:

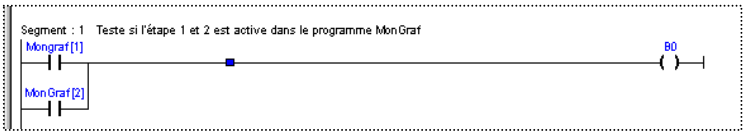
MonGraf : Array [0..9] of Boolean;
TmonGraf : Array [0..9] of Integer;

MonGraf représente les 10 états des étapes du programme MonGraf.
TmonGraf représente les 10 temps d'activation des étapes du programme MonGraf.

Ex: Raz d'un Grafcet :

For I:=0 **to** 9 **do** MonGraf[I]:=False;
*{** a l'aide d'une boucle For le programme force toutes les étapes du Grafcet à 0 **}*

Il est possible dans un schéma à contacts de tester si l'étape d'un Grafcet est active.



Ici un OU est effectué sur l'état des 2 étapes du programme MonGraf.

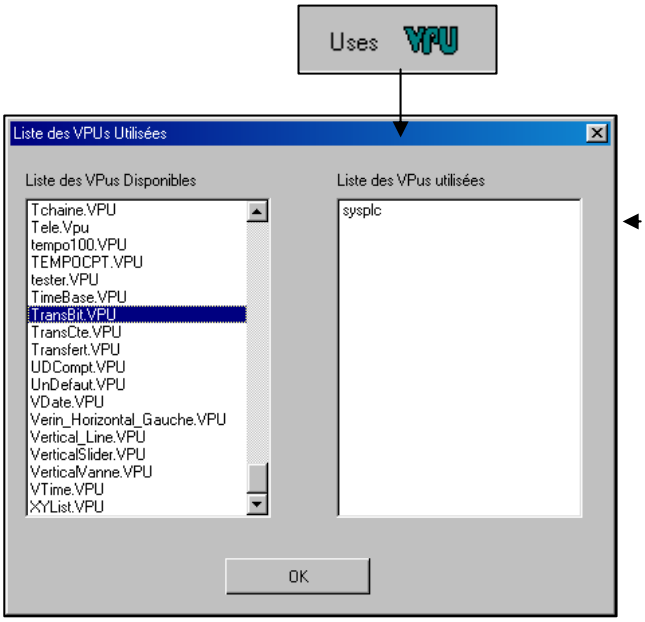
Dans une transition de Grafcet, l'on peut avoir une équation testant le temps d'activation d'une étape.

If TmonGraf[2]>100 **Then** Mongraf[3]:=True;

Si l'étape 2 est active plus de 10 secondes, l'étape suivante 3 est forcée, mettant automatiquement l'étape à False puisque ces 2 Etapes sont consécutives.

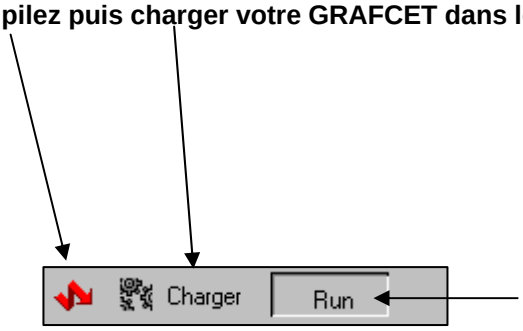
Utilisation de fonctions externes dans un Grafcet.

Le grafcet peut faire appel à des fonctions externes que vous auriez écrites dans une librairie.
A fin que le compilateur puisse trouver ces fonctions et procédures, vous devez spécifier les librairies que vous voulez utiliser.
Ceci à l'aide du bouton :



Double cliquez dans la liste de gauche sur la librairie que vous voulez utiliser.
Cette librairie se place dans la liste de droite.
Ainsi vous pourrez utiliser toutes les fonctions contenues dans cette librairie dans vos équations de transitions ou d 'étapes.

Compilez puis charger votre GRAFCET dans le moteur temps réel de la même façon qu'en LADDER.



Run/Stop de tous les programmes du moteur temps réel

Introduction au langage pascal

Description

Arsoft avec Visual I/O qui est le complément indispensable de Visual PLC pour les applications d'interface Homme/Machine à une très grande expérience du langage Pascal. Le langage décrit dans la norme IEC1131 est très proche de la syntaxe du langage Pascal celui-ci existant bien avant cette norme. C'est pour une raison d'homogénéité qu'Arsoft a choisi ce langage plutôt qu'une définition venant d'une norme. Le programmeur Visual PLC pourra migrer très simplement vers Visual I/O après avoir programmé en Visual PLC. De plus l'interconnexion avec le système Windows devient de plus en plus demandé et doit mettre en œuvre des mécanismes dépassant ceux prévus par la norme. Nous trouvons bien entendu les instructions de base (affectations, Boucle For, If/Then/Else, Case etc.) mais aussi les possibilités de transtypage, d'accès aux DLL extérieures et bien plus encore.
Le compilateur Pascal de Visual PLC à été étudié pour les problèmes d'automatisme. Ce compilateur n'est pas un compilateur Bureautique ! Le compilateur Arsoft met en œuvre des algorithmes puissants pour diminuer et optimiser le temps d'exécution des programmes.

Le compilateur de Visual PLC permet :

L'optimisation du code généré
La compilation au vol sans état d'attente.
La compilation incrémentale.
La compilation directe sans transformation du code source.
La compilation monopasse.
La compilation avec infos de débuggage.
Le découpage du programme en module VPUs directement exécutable par le moteur Temps réel.

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 PLC.
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 Element est passée à la procédure. Cette Variable ne peut être seulement lue.
Dans le cas de la procédure MonProc2 la variable Element 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) ;
```

Ecriture de modules fonctionnels

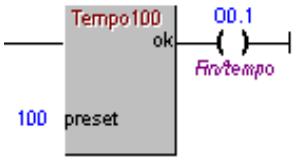
L'éditeur de modules fonctionnels, permet selon une philosophie particulière d'écrire des blocs de programmes qui peuvent êtres assimilés à des procédures avec passage de paramètres.
Pour écrire des modules fonctionnels vous pourrez utiliser toute la force du langage pascal ainsi que faire appel à des procédures et fonctions que vous aurez écrites ou venant de l'extérieur.
Allant des équations simples aux imbrications complexes, Visual Pascal est un véritable langage comparable à ceux utilisés pour le développement d'exécutables sous Windows.

Cependant le compilateur Arsoft dans la version Visual PLC est très orienté automatisme, optimisant les traitements booléens et le type de variables pouvant être passées (exemple un bit dans un mot).

Philosophie d'un module fonctionnel

Un module fonctionnel est une boite noire ayant une fonction particulière. Cette boite reçoit des paramètres en entrée et en sortie ou n'en reçoit pas.
Si des paramètres sont nécessaires, ils peuvent êtres en entrée ou/et en sortie du module.

Prenons le cas de la temporisation 100ms livrée. Sa représentation graphique est :

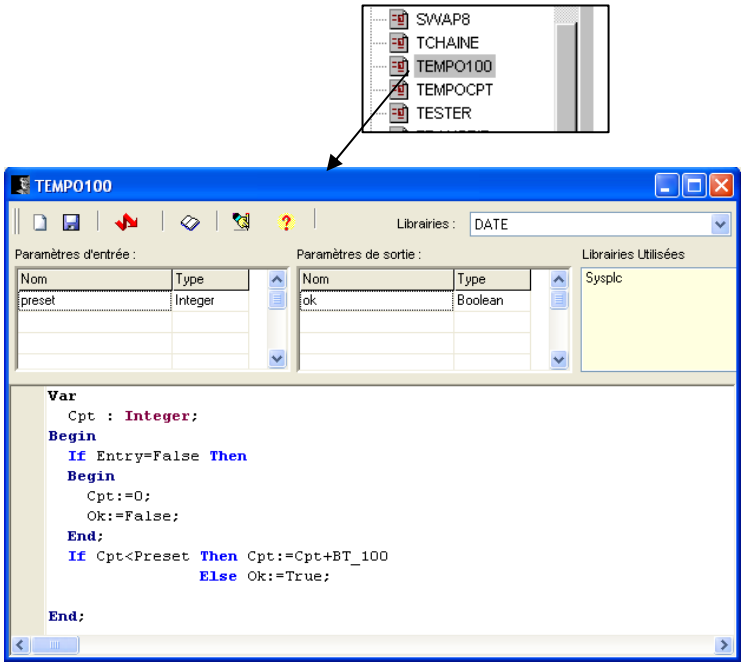


Nous nous apercevons qu'il y a 2 paramètres :

1 paramètre d'entrée **preset** se situant **à gauche** du module.
1 paramètre de sortie **ok** se situant **à droite** du module.

Il apparaît que le paramètre d'entrée est du type numérique et celui de sortie du type booléen.

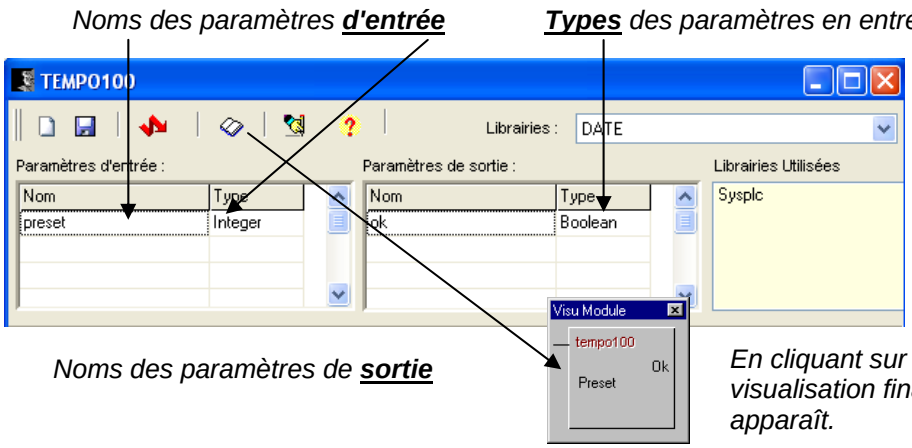
Appelons l'éditeur de modules fonctionnels pour visualiser le code source de cette temporisation :
Double-cliquons sur *Tempo100* dans la rubrique Modules de l'explorateur:



Programmation dans un module fonctionnel

Examinons le code source du module temporisation 100 ms :

La déclaration des paramètres :



Dans les 2 grilles, inscrivez les noms des paramètres en entrée et en sortie. Ici preset est un paramètre d'entrée de type Integer et Ok un paramètre de sortie de type Boolean.

Lorsque vous inscrivez un nouveau paramètre en entrée ou en sortie, le type Integer s'affiche automatiquement. Pour changer de type double cliquez sur la cellule faisant ainsi défiler les nouveaux types possibles.

Les types possibles en Entrée : Integer, Boolean, String et Real
Les types possibles en Sortie : Integer, Boolean, String, Real et ALL

Integer peut recevoir des variables et des constantes numériques de type Integer, Word, Byte et éventuellement Real.
Si un paramètre d'entrée est de type Integer et qu'il reçoit une variable ou une constante de type Real, un arrondi est fait automatiquement en réception sur cette valeur.

Boolean peut recevoir des variables booléennes ou des bits d'un entier (Integer, Word).
Exemple : Cde ou I0.0

String peut recevoir des variables chaîne de caractères ou des constantes en chaînes de caractères.

Real peut recevoir des variables numériques de tout type en entrée. Ces variables seront de toute façon traduite en valeurs flottantes. En sortie une variable flottante (Real) sera demandée obligatoirement.

ALL peut recevoir tout type de variables seulement en sortie. Cette variable fera l'objet d'un transtypage par le module. Le transtypage sera abordé plus loin.

Test de l'équation en schéma à contact appelant le module fonctionnel

Il apparaît intéressant de pouvoir tester les conditions d'appel d'un module fonctionnel. En d'autres termes connaître le résultat du schéma à contact accroché au module.
La variable prédéfini **ENTRY** de type Boolean reflète l'état appelant.

Si **ENTRY** vaux **True**, cela signifie que le **schéma** à contact accroché au module est **passant**.

Analysons le code source de la temporisation :

```
Var
  Cpt : Integer;
Begin
  If Entry=False Then          Si l'appel est faux(0),le bit de sortie
  Begin                        est remis à 0 ainsi que le compteur
    Cpt:=0;                    interne.
    Ok:=False;
  End;
  If Cpt<Preset Then Cpt:=Cpt+BT_100      Si le compteur interne est
                Else Ok:=True;            inférieur à la valeur
End;                                  passée on incrémente le
                                      Compteur interne sinon on
                                      Passe le bit de sortie à
                                      True (1).
```

Nous pouvons voir que le module à besoin d'une variable interne sauvegardée. Cette variable est déclarée avant le Begin général. Dans le cas d'un module fonctionnel, toute variable déclarée de cette façon est sauvegardée quand le programme quitte le module.
Nous avons la possibilité de déclarer en locale au module tout type de variables (tableaux, chaînes etc.).

Variables Système

ENTRY	(Boolean)	Indique l'état du schéma à contact appelant.
BT_100	(Integer)	Passe à 1 tout les 100ms et sur un tour de cycle.
START_PROG	(Boolean)	Indique le premier tour de cycle du programme.

Attention dans un module fonctionnel, les variables locales au module ne sont pas initialisées.
Il est possible d'initialiser ces variables cette grâce à la variable Start_Prog.

Exemple: **If** Start_Prog **Then** Cpt:=0;

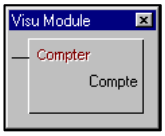
Sortie immédiate d'un module fonctionnel
La commande **EXIT** permet la sortie immédiate d'un module sans continuer l'exécution du reste du programme du module.

Exemple :

```
      If Entry=False Then
            Begin
                Cpt:=0;
                Ok:=False;
                Exit;   {Sortie immédiate du module ici}
            End;
```

Exemples de modules fonctionnels

Compteur Simple :
Considérons un module ayant un seul paramètre en sortie de type Integer nommé compte et que nous voulions compter des impulsions sur le schéma à contact appelant.



Si le programme était écrit ainsi

```
Begin
  If Entry Then Compte:=Compte+1;
End;
```

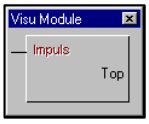
Le mot compte serai incrémenté autour de cycle. Il faut donc insérer un front montant en déclarant un Boolean servant de bit de front.

```
Var
  Front : Boolean;      {Bit de front interne sauvegardé}
Begin
   If Entry and Not(Front) Then {Si entrée est passante et pas front alors
                                 incrément.}
   Begin
      Compte:=Compte+1;
      Front:=True;
   End;
   If Not(Entry) Then front:=False; {Si plus entrée passante bit de front passe à
                                     False.}
End;
```

Impulsions :
Considérons un module ayant un seul paramètre en sortie de type Boolean nommé Top et que nous voulions envoyer des tops tous les 100ms si le schéma à contact appelant est vrai.

```
Begin
  If Entry Then   //Si schéma passant on traite le bloc en dessous
  Begin
    If BT_100=1 Then Top:=True //Si Bt_100=1 le bit de sortie passe à True (1)}
                Else  Top:=False; //sinon on passe le bit de sortie à False (0).}

   End Else Top:=False;
End;
```

Variante / Optimisation:




```
Begin
    If Entry Then Top:=(BT_100=1)
             Else Top:=False;
End;
```

Il faut comprendre : Top étant un booléen, nous affectons le résultat du test (BT_100=1) qui est vrai ou faux à ce booléen.

Bloc ET 2 entrées :
Considérons un module ayant 2 paramètres en entrée nommés E1 et E2 et 1 paramètre en sortie nommé Res. Tous les paramètres sont de type Boolean.


```
Begin
  If Entry Then  Res:=E1 and E2
           Else  Res:=False;
End;
```

Transfert de tables et transtypage

Il est souvent nécessaire dans un module fonctionnel de transférer des mots consécutifs dans d'autres mots étant aussi consécutifs. Ou tout simplement remettre à zéro une table entière.

Prenons comme exemple le transfert d'une constante dans un tableau de mots (Integer).

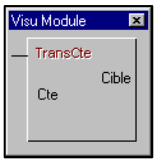
Considérons que vous ayez déclarez un tableau d'integers dans les variables globales. Ce tableau est déclaré comme suit :

```
Tmots : Array [0..12] of Integer;
```

Construisons un module fonctionnel capable de transférer une constante dans ce tableau.

Déclarons un paramètre d'entrée appelé **Cte** de type **Integer** recevant la constante à transférer.
Déclarons un paramètre de sortie appelé **Cible** recevant le tableau. Le problème est de choisir un type pour ce tableau. Ce n'est pas un simple Integer puisque c'est un tableau.
C'est là qu'intervient le type **ALL**. Cette déclaration qui est sans type permet de passer n'importe qu'elle variable à un module fonctionnel.

Le programme peut être le suivant :




```
Var
  I : Integer;
Begin
   If Entry Then
   Begin
     For I:=0 to 12 do Cible[I]:=Cte;
    End;
End;
```

Explications :
Une boucle **For** balaie tout le tableau passé. Comme Cible est du type **ALL** il peut être adressé de toutes les façons (Tableau de Boolean, Real, chaîne etc.). A la compilation du module, le compilateur ne sait pas quel est son type, il le devine par la syntaxe employée et l'affectation qu'il reçoit.

Ici puisqu'il y a des crochets ([I]) le compilateur devine que c'est un tableau, oui mais de quoi ?
Le compilateur regarde alors le type de variable que reçoit ce tableau, il voit que c'est un Integer, il en déduit alors que c'est un tableau Integer. Cette opération s'appelle un transtypage simplifié.

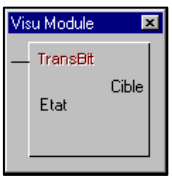
Transfert d'un état dans un tableau

Nous déclarons un tableau de Boolean dans les variables Globales :

```
Tbits : Array [0..12] of Boolean;
```

Le programme est très approchant à celui décrit ci-dessus. Simplement le paramètre d'entrée est un Boolean et le tableau de sortie est un tableau de Boolean. Le transtypage par le compilateur est ici du type Boolean

Le programme peut être le suivant :




```
Var
  I : Integer;
Begin
   If Entry Then
   Begin
     For I:=0 to 12 do Cible[I]:=Etat;
    End;
End;
```

Transfert d'une table dans une autre table

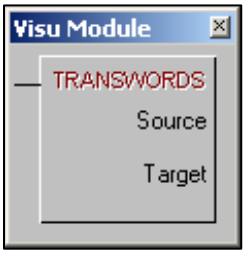
Considérons que dans les variables globales les tableaux ci-dessous soient déclarés:

```
TabSour     : Array [1..20] of Integer;
TabDesti    : Array [0..19] of Integer;
```

Les 2 tableaux ont la même dimension mais ne commencent pas au même indice.

Un module fonctionnel permettant de transférer une table dans une autre doit avoir 2 paramètres de type ALL donc obligatoirement en sortie. Déclarons Source et destination comme paramètres de sortie de type ALL.

Le paramétrage en schéma à contact pourra être
de ce type en passant le premier élément de TabSour et de TabDesti.



Le programme dans le module fonctionnel est :

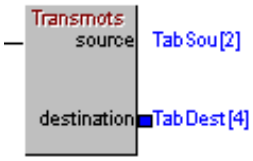
```
Var
  i : Integer;
Begin
  If Entry=False Then Exit;
  For I:=1 To 4 Do
   TArrayInt(Target)[I]:=TArrayInt(Source)[I];
End;
```

Comme Source et Target sont tous les deux de type ALL, le compilateur a besoin de connaître un des types pour affecter l'autre. D'où **TArrayInt (Source[I])** indiquant au compilateur (celui-ci le croyant sur parole) que source est un tableau d'Integer et de ce fait que TARGET est du même type. Ici le programme transfert 4 mots de TabSour vers TabDesti. Ceci est un transtypage classique de variables.

Déjà déclaré dans Sysplc.Pas et utilisé pour les transtypages

```
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;
```

Autre type de paramétrage en schéma à contact



Ici le module fonctionnel décrit ci-dessus transférera 4 éléments consécutifs démarrant de TabSour[2] dans 4 éléments consécutifs démarrant de TabDesti[4].

Note Importante :
Lorsqu'une variable de type ALL est utilisée dans un module fonctionnel, le compilateur en cas de transtypage sous forme de tableaux considère que celui-ci démarre à l'indice 0
(Array [0..X] of Y).

Autre possibilité de transfert à grande Vitesse l'instruction MOVE :
Le programme de notre module fonctionnel peut être écrit ainsi :

```
Begin
    If Entry Then MOVE(Source,Destination,16);
End;
```

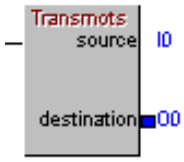
L'instruction **Move** transfère le contenu de la variable Source dans la variable Destination d'un nombre d'octets spécifiés. Ici 4 integers=16 octets.

Transfert de variables

Considérons que dans les variables globales les tableaux ci-dessous existent

```
I0,I1,I2,I3       : Integer;
O0,O1,O2,O3       : Integer;
```

Le module fonctionnel décrit avant permet de transférer des valeurs d'un tableau dans un autre tableau. Cependant comme un tableau est constitué de valeurs consécutives, il est possible grâce à ce module fonctionnel de transférer d'autres valeurs consécutives n'appartenant pas à un tableau.



Dans ce cas le paramétrage dans le schéma à contacts peut aussi être le suivant. Puisque I0,I1,I2,I3 sont déclarés consécutivement, il est possible au module fonctionnel de les prendre comme un tableau afin de les transférer dans d'autres variables consécutives comme O0,O1,O2,O3.

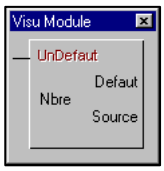
Test de bits dans des variables consécutives

Considérons un module permettant de tester si une des entrées du process est à 1. Les entrées physiques du process sont contenues dans les variables I0,I1,I2,I3 de type Integer;

Le module pourrai être le suivant :

En entrée le nombre de mots scruté. Type Integer.
En sortie un bit passant à 1 si un des bits d'entrée est détecté à un. Type Boolean.
En sortie le premier mot d'entrée à scruter type ALL.

```
Var
  I : Integer;
Begin
   Defaut:=False;
   For I:=0 to Nbre do
   If TArrayInt(Source)[I]<>0 Then Defaut :=True;
 End;
```



Décalage de bits dans des mots consécutifs.

Considérons un module permettant de décaler vers la gauche tous les bits de mots consécutifs.
Le programme pourrait être le suivant :

```
Var
 I    : Integer;
 Front : Boolean;
Begin
  If Entry and Not(Front) Then
  Begin
    For i:=3 Downto 0 do
    Begin
       TArrayInt(Source)[I]:=TArrayInt(Source)[I] Shl 1;
        If I>0 Then
         If Source[I-1].31 Then Source[I].0:=True;
    End;
    Front:=True;
  End;
  If Entry=False Then Front:=False;
End;
```

Explications :

```
If Entry and Not(Front) Then
```
Permet de faire un seul décalage quand le schéma à contacts connecté est passant.

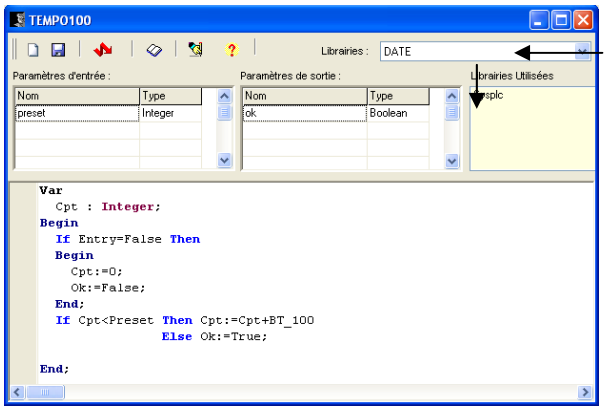
```
For i:=3 Downto 0 do
```
Boucle partant de 3 allant vers 0. Démarrage sur le dernier Mot pour aller au premier Mot.

```
Source[I]:=Integer(Source[I]) Shl 1;
```
Tous les bits du mot en cours sont décalés de une position vers la gauche.

```
If I>0 Then
     If Source[I-1].31 Then Source[I].0:=True;
```
Si le pointeur n'est pas sur le premier mot, nous prenons le dernier bit du mot précédent qui se reporte dans le premier bit du mot en cours.
Ici l'on traite des mots de 32 bits.

Appel de fonctions et de procédures dans une librairie VPU.

Nous verrons par la suite qu'il est possible de créer ses propres fonctions et procédures dans des librairies VPU.
Dans un module fonctionnel il est possible d'utiliser des procédures et des fonctions localisées dans des librairies VPU.
La fonction **Sound** permettant de jouer une musique par la carte sonore de votre PC est programmée dans la librairie **SysPLC**.
Pour qu'un module fonctionnel puisse utiliser cette procédure, il faut indiquer le nom de la librairie VPU contenant la procédure.



Pour indiquer au compilateur de modules fonctionnels d'inclure les procédures et fonctions d'une librairie, amenez par drag & drop le nom de la librairie souhaitée dans la liste en dessous.
Ici la procédure Sound de la librairie Sysplc est utilisée et acceptée par le compilateur.

Compilation d'un module fonctionnel

La compilation s'effectue par clic sur le bouton ZigZag. Le compte rendu de cette compilation est écrit en rouge dans la zone statuts en bas de l'écran.

Si une erreur est détectée, une vidéo inverse indique le point d'arrêt du compilateur trouvant l'erreur. Si aucune erreur n'est détectée le texte 'Compilation Ok' est inscrit en bas et le fichier de nom du module et d'extension **.VPU** est créé et stocké sur disque.

Jeu d'instructions du langage pascal pour Visual PLC

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 à0`
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

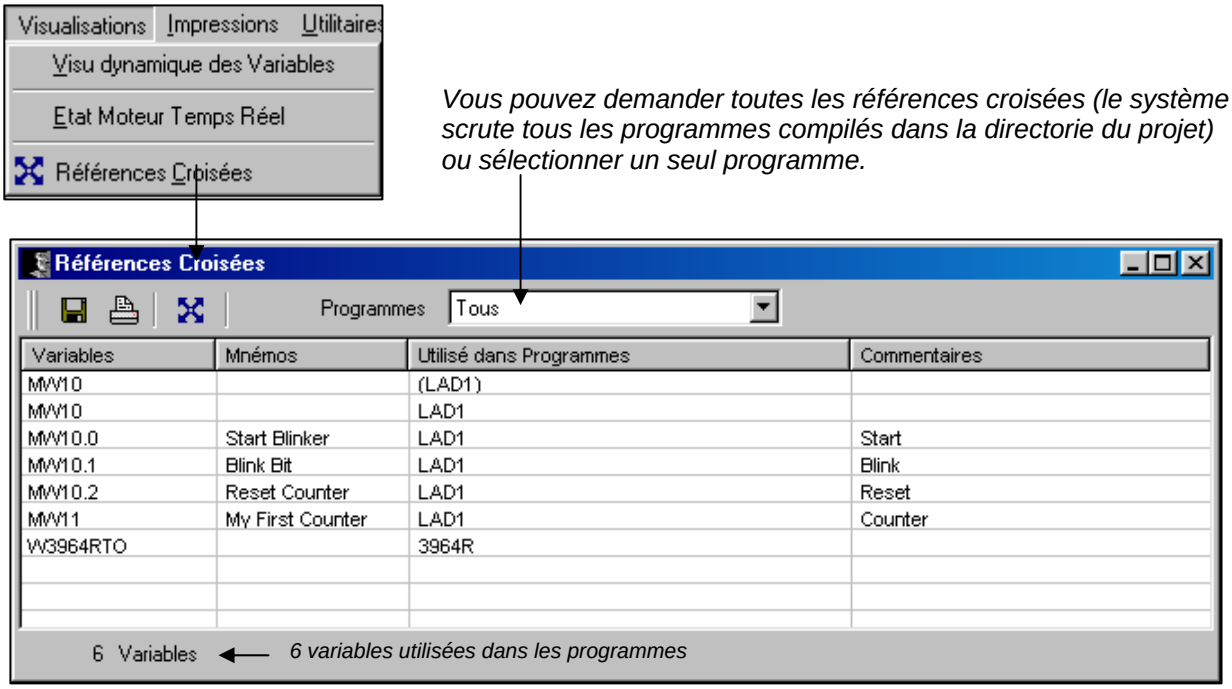
```
V:=Mem($C800,0);          // ** lecture d'un byte *
```

En écriture affecte d'une valeur une adresse mémoire

```
Mem($C800,0):=255;        // ** écriture d'un byte
```

Références croisées

Vous pouvez obtenir la liste des bits, constantes et modules fonctionnels utilisés dans tous les programmes compilés (Ladders, Grafcets, programmes etc..), cliquez dans le menu principal Visualisations/ Références Croisées.



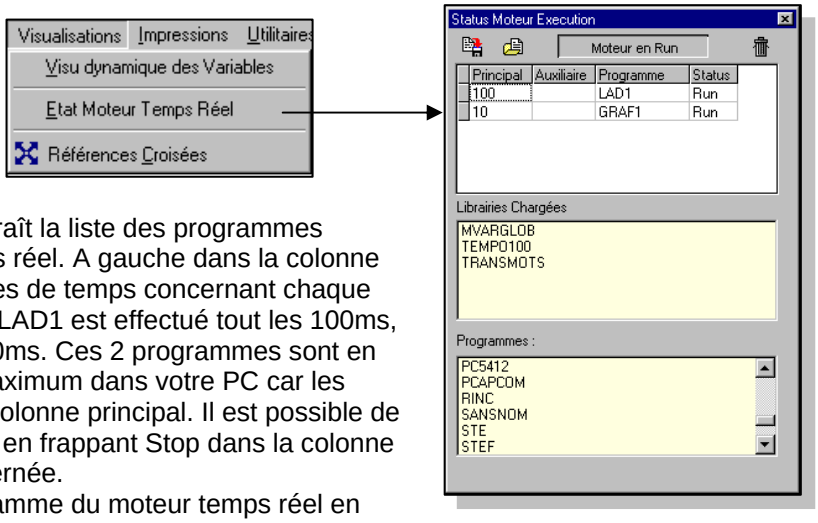
Explication :

(LAD1) : Bobine ou activation par programme.
LAD1 : Utilisé en contact ou dans une équation ou expression.

Note : Les références croisées sont effectuées par le compilateur (à la compilation des programmes).

Statuts du moteur temps réel

Les programmes sont chargés dans le moteur temps réel par les boutons 'chargement' dans les différents éditeurs (Ladder, Grafcet).
Cependant il est possible de charger le moteur temps réel manuellement. Surtout dans le cas de programmes écrits en langage évolué par l'éditeur 'librairies Vpu'.



Dans la grille supérieure apparaît la liste des programmes présents dans le moteur temps réel. A gauche dans la colonne Principal sont affichés les bases de temps concernant chaque programme. Ici le programme LAD1 est effectué tout les 100ms, GRAF1 est effectué tout les 10ms. Ces 2 programmes sont en Run et effectués en priorité maximum dans votre PC car les bases de temps sont dans la colonne principal. Il est possible de passer un programme en stop en frappant Stop dans la colonne statuts et dans la cellule concernée.
Vous pouvez effacer un programme du moteur temps réel en cliquant sur le nom du programme puis en frappant la touche <Suppr>.

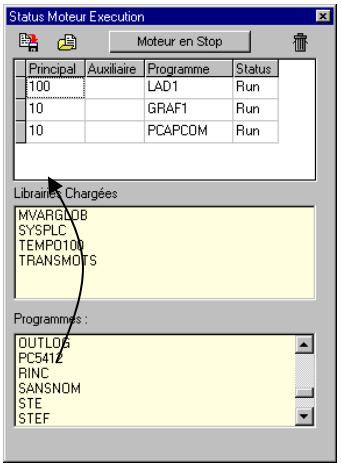
Pour changer la base de temps d'un des programmes, cliquez dans la cellule à gauche du nom du programme concerné puis frappez la nouvelle base de temps souhaitée. Cette base de temps ne peut être qu'un multiple 10ms.

Ajout d'un programme dans le moteur Temps Réel

Pour ajouter un programme quelconque dans le moteur temps réel, amenez le nom du programme concerné de la liste inférieure par Drag & Drop et déposez ce nom dans la grille supérieure. Le nom programme vient s'inscrire à la suite des autres programmes.

En cliquant sur ce bouton, tous les programmes présents dans le moteur temps seront effacés.

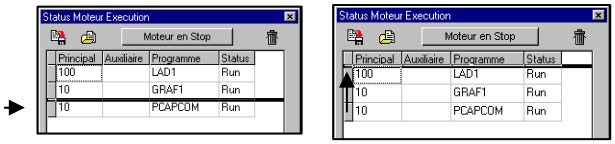
Ce bouton permet de mettre en Run ou en Stop tous les programmes du moteur Temps réel.



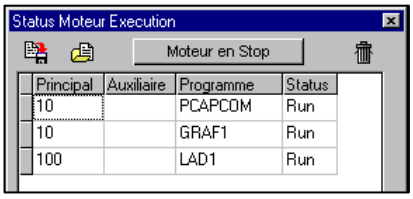
Ordre d'exécution des programmes dans le moteur temps réel

Il est possible d'organiser l'ordre d'exécution des programmes dans le moteur temps réel. Pour cela déplacez le programme concerné dans la liste supérieure vers l'emplacement désiré.

Cliquez sur la cellule grise à gauche du nom du programme concerné. Puis déplacez vous vers la position concernée. Une barre épaisse apparaît.



Lorsque vous relâchez le bouton gauche de la souris, une inversion des programmes source et cible est effectuée. Cependant à ce stade le moteur temps réel n'a toujours pas été affecté.
Il faut **sauvegarder** la liste de ces programmes puis la **recharger** dans le moteur à l'aide des 2 boutons dans la partie supérieure.



Sauvegarde de la configuration du moteur temps réel

Pour sauvegarder la liste des programmes visualisés dans la fenêtre cliquez sur le bouton 'Sauvegarde configuration actuelle'.

La liste des programmes est sauvegardée dans le fichier texte PROG.PLC. Ce fichier permettra un chargement automatique du moteur temps au Boot de Windows.

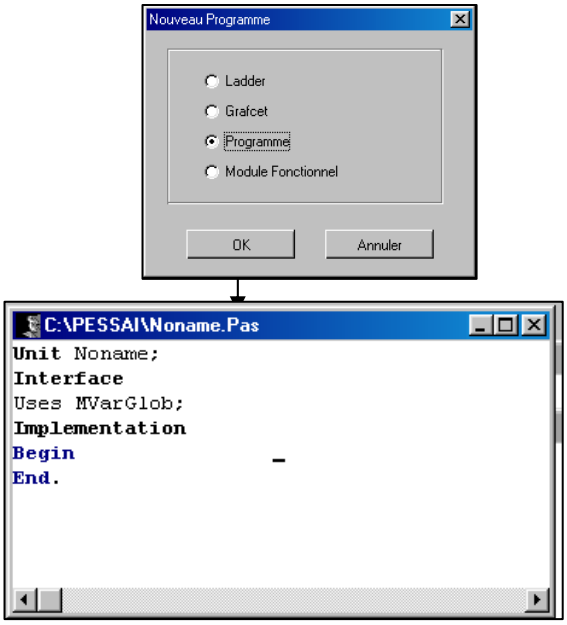
Chargement d'une configuration dans le moteur temps réel

Pour charger la dernière configuration dans le moteur temps réel cliquez sur ce bouton. Celui-ci permet de charger le moteur avec les programmes listés dans le fichier PROG.PLC.

Ecriture des programmes cycliques en langage Pascal

Il est possible sous Visual PLC d'écrire un programme cyclique (VPU) qui sera chargé dans le moteur temps réel au même titre qu'un programme écrit en Ladder ou en Grafcet.

Pour cela appelez l'éditeur de programme par clic sur le bouton nouveau et sélection programme



L'éditeur de programmes et librairies apparaît. Un masque de librairie est proposé

Cet 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 ***}
```

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 le moteur temps réel.
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 des boutons dans la barre d'outils :

Programmes cycliques avec interface

Un programme normalement ne peut avoir de variables ou de procédures à échanger, puisqu'il utilise ces procédures, fonctions ou variables dans des librairies. Visual PLC permet de mixer programmes et librairies.

Considérons un programme qui incrémente un mot, et qu'avec ce mot nous voulions qu'il appartienne au programme mais aussi qu'il soit accessible à d'autre programmes.

Examinons ce code :

```
Interface
Var
  ID0 : Integer;        {** variable accessible de l'extérieur **}
Implementation
{*** Fonction interne au programme ***}
Function Inc( Value : Integer) : Integer;
Begin
  Result:=Value+1;
End;
{** Corps du programme ***}
Begin
  ID0:=Inc(ID0);
End.
```

Nous pouvons apercevoir que la clause Uses à disparue, car nous n'utilisons pas de variables globales ni de fonctions contenues dans d'autres VPU.
La clause **INTERFACE** est apparue. Cette clause permet d'indiquer au compilateur que nous voulons mettre à disposition des autres programmes des variables, des procédures ou des fonctions. Dans notre exemple, nous voulons mettre simplement à disposition une variable qui s'appelle IDO de type Integer.
La clause **IMPLEMENTATION** permet de délimiter la fin d'interface et démarrer le code et les variables propres au programme mais non accessibles à d'autres programmes.
La fonction **Inc** permet d'incrémenter une variable. Cette fonction ne pourra être utilisée que dans le programme lui-même car elle est contenue dans la partie implémentation.
Enfin nous arrivons sur le corps du programme qui incrémente au tour de cycle la variable ID0.

La variable ID0 est maintenant accessible dans le tableau de visualisation de variables au même titre que toutes les variables globales de Mvarglob.VPU mais seulement quand le programme noname3 est chargé dans le moteur temps réel.

Voilà un mixage de programme et de librairie. En général une librairie ne contient pas de programme et celui-ci ne fait pas office de librairie.

Ce cas de figure peut être intéressant pour avoir un contrôle sur le déroulement interne, puisqu'il offre une interface ID0 qui pourrait être une commande selon son état.

Si nous voulons que les autres programmes est accès à la fonction INC, le programme serai écrit ainsi :

```
Interface
Var
  ID0 : Integer;        {** variable accessible de l'extérieur **}
Function Inc( Value : Integer) : Integer; //cette fonction est maintenant
                                          // accessible aux autres
Implementation
{*** Fonction interne au programme ***}
Function Inc( Value : Integer) : Integer;
```

Le fait de réécrire la déclaration de la fonction Inc, la rend accessible à l'extérieur à d'autres programmes.

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 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.

Le code inscrit dans le corps principal sera effectué une seule fois lorsque :

- La librairie sera chargée et que le moteur temps réel sera mis en route.
- A chaque redémarrage du moteur Temps (passage en stop puis en Run).

Cette fonctionnalité pourra être utilisée pour initialiser des variables ou des équipements avant le démarrage effectif de l'application.

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;

Procedure Initial;    {procedure 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 est 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
   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.

Appel de procédures et de fonctions d'une DLL

Visual PLC 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 PLC 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 PLC 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 PLC 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 Respecter 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 en 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 PLC qui est Bip et qui correspond à MessageBeep dans user32.dll.

Localisation d'une DLL.

Il est possible en l'absence de chemin que le moteur temps réel de Visual PLC 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 95/98). Le moteur temps réel 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';
```

Exemple d'affectation de variables Visual PLC dans une DLL externe

Considérons le programme Pascal suivant implémenté dans une DLL de nom MyDLL.
Ce programme permet de forcer à 1, 10 Integers dans Visual PLC.

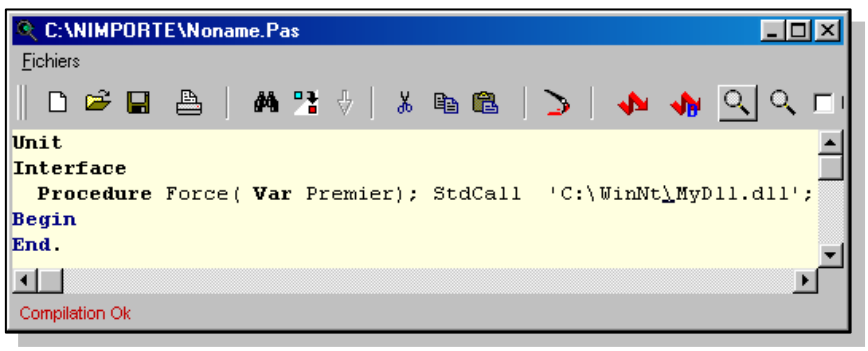
Exemple en Delphi ou Visual Pascal :

```
Library Mydll;
{$C PRELOAD MOVEABLE DISCARDABLE}

 {******** Met à 1 tous les Integer dans Visual I/O ou PLC ***********}
Procedure Force( P : Pointer); StdCall;
Var
   I        : Integer;
   PTInt    : ^Integer;   //*** Pointeur sur Integer
Begin
  PTInt:=P;  //**** transfert le Pointer recu
  For I:=1 To 10 do
  Begin
     PTInt^:=1;   //*** Met le mot Visual I/O ou Visual PLC à 1
     Inc(PTInt);  //*** Passe au mot suivant
  End;
End;

{******* Exports **************}
 Exports
    Force Index 1;
{******** Your Initialisation ***********}
Begin
End.
```

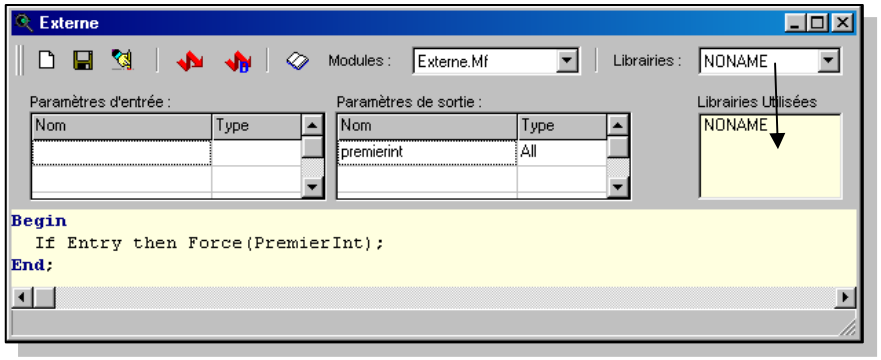
Afin d'appeler la procédure Force dans cette DLL, il faut déclarer le prototype dans une librarie VPU de Visual PLC.

Exemple :

La librairie Noname.pas permet l'interfaçage avec la procédure localisée dans MyDll.com.

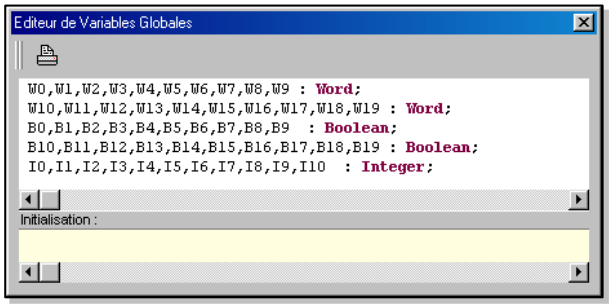


Créons maintenant un module fonctionnel conditionnant l'appel à cette procédure (Force).

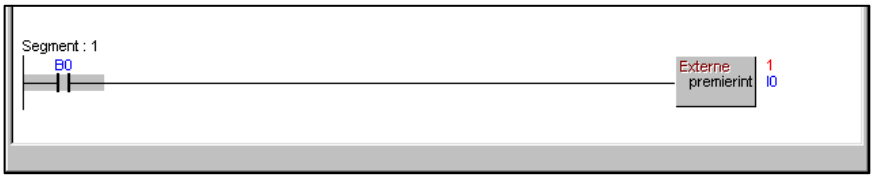


Le type **All** permet de recevoir une adresse (pointeur) dans le module fonctionnel. Cette adresse est retransmise à la procédure Force de notre DLL.

Déclarons au moins 10 integers (I0 à I10) nous permettant de faire ce test.



Plaçons ce module fonctionnel dans un schéma à contacts.



En forçant le bit B0 le module fonctionnel est validé en interne et appel la procédure force lui passant l'adresse du premier mot (I0).
Il est possible de faire de même avec n'importe quelle variable de Visual PLC. Cette variable peut être un élément de tableau par exemple.

Lancement du moteur temps réel en automatique

Après mise au point à l'aide de l'éditeur Visual PLC, les programmes sont sauvegardés dans un fichier nommé PROG.PLC en cliquant sur le bouton 'Sauvegarde Configuration Actuelle' (voir chapitre 'Statuts du moteur temps réel'.

Ce fichier texte contient la liste de tous les programmes qui doivent êtres chargés par le moteur. Ce fichier ne contient pas les librairies utilisées par ces programmes.
Si vous devez changer de machine vous devez transporter :

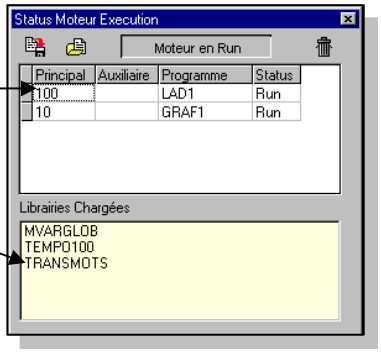
Les fichiers programmes (.VPU)
Les fichiers librairies utilisées (.VPU).

Exemple :

Le moteur utilise les programmes :
LAD1.VPU et **GRAF1**.VPU

Ainsi que les librairies
MVARGLOB.VPU,
TEMPO100.VPU,
TRANSMOTS.VPU

Vous devez alors transporter ces 5 fichiers sur la machine cible.



Pour démarrer le moteur temps réel, lancez le programme PLCRUN.EXE se trouvant dans le répertoire API32. Ou PLCRUN *NomDeConfiguration.PLC*

Ce programme cherche le fichier PROG.PLC dans la directorie courante.
Si le fichier est absent PlCRun le cherche dans le projet courant.

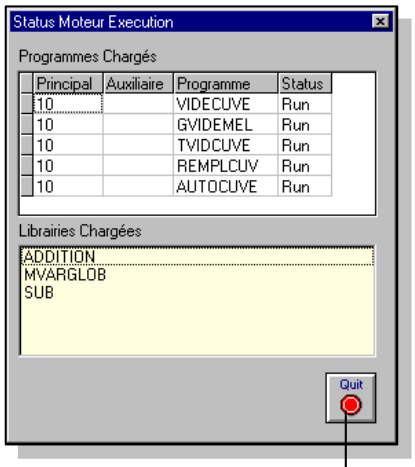
Le moteur temps réel contrairement à l'éditeur Visual PLC, nécessite une clef de protection. Cette clef peut être située sur le USB de l'ordinateur ou embarquée sur la carte de communication. Dans le cas d'une clef USB le driver est automatiquement installé lors de l'installation générale de Visual PLC.

Le programme PLCRUN.EXE peut être lancé automatiquement dans le groupe de démarrage de Windows. Après lancement une icône se place dans la barre de taches de Windows.

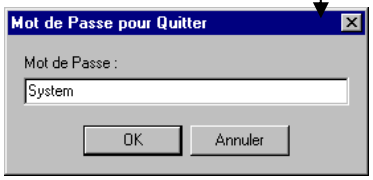
En cliquant sur cette icône une fenêtre apparaît, listant tous les programmes chargés par le moteur.

Visualisation des programmes chargés

En cliquant sur l'icône dans la barre de taches de Windows une fenêtre apparaît :

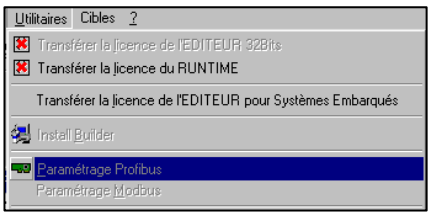


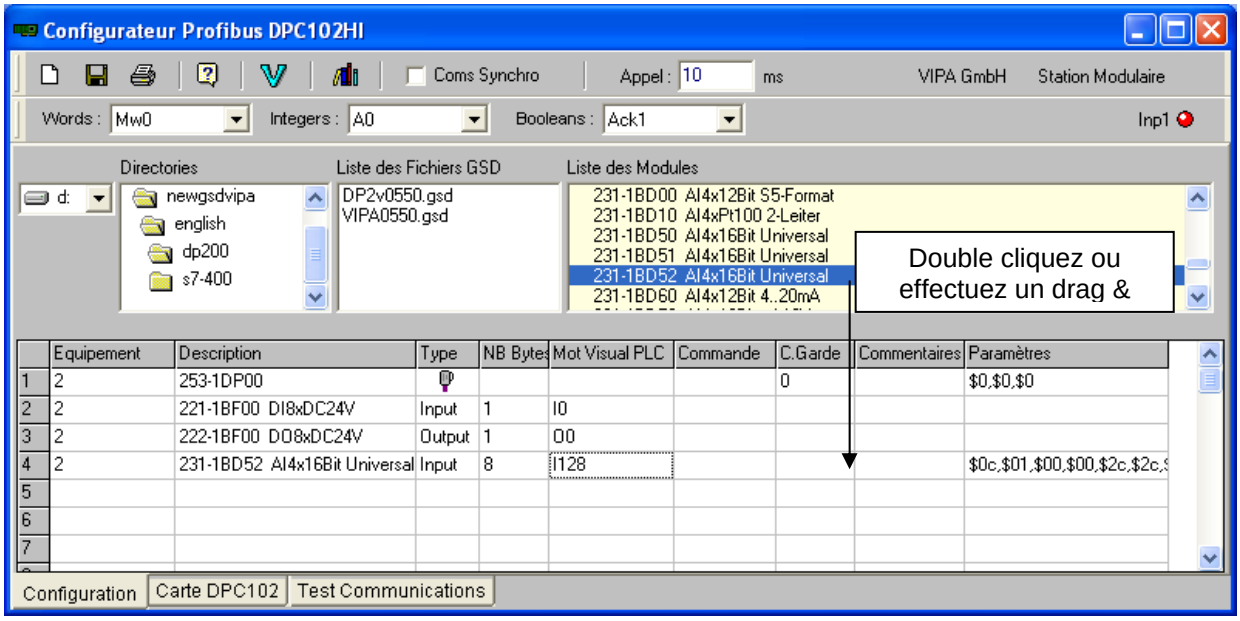
En cliquant sur le bouton 'Quit', il est possible d'arrêter le moteur Temps réel en frappant le mot de passe **'System' ou '456789'**.



Configurateur Profibus pour carte DPC102HI D'ARSOFT.

Lancez l'utilitaire **SERVDP.EXE**, une fenêtre apparaît :





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)

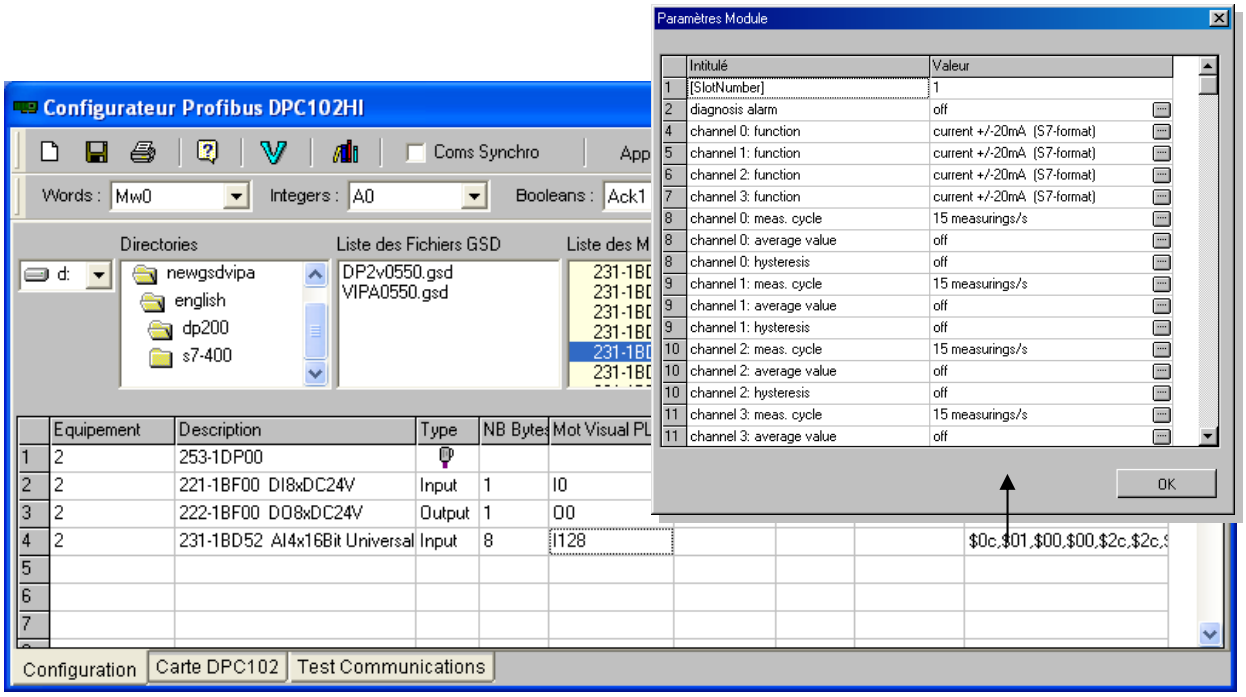
Commentaires : Vos commentaires.
Paramètres : Permet de visualiser ou de modifier les paramètres d'initialisation de la station et des modules.

☞ **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.

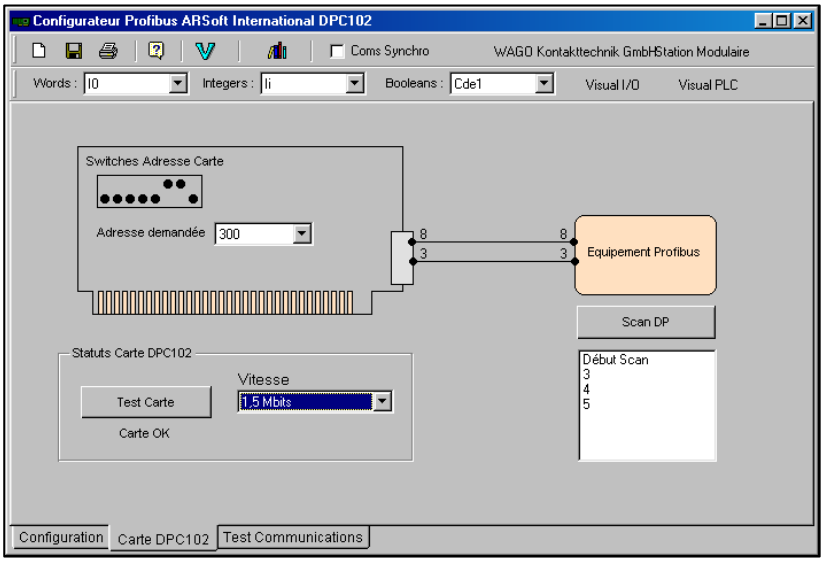
Réglage des paramètres de l'esclave

Double-cliquez dans la **colonne paramètres** pour régler chaque paramètres du module concernée. 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.

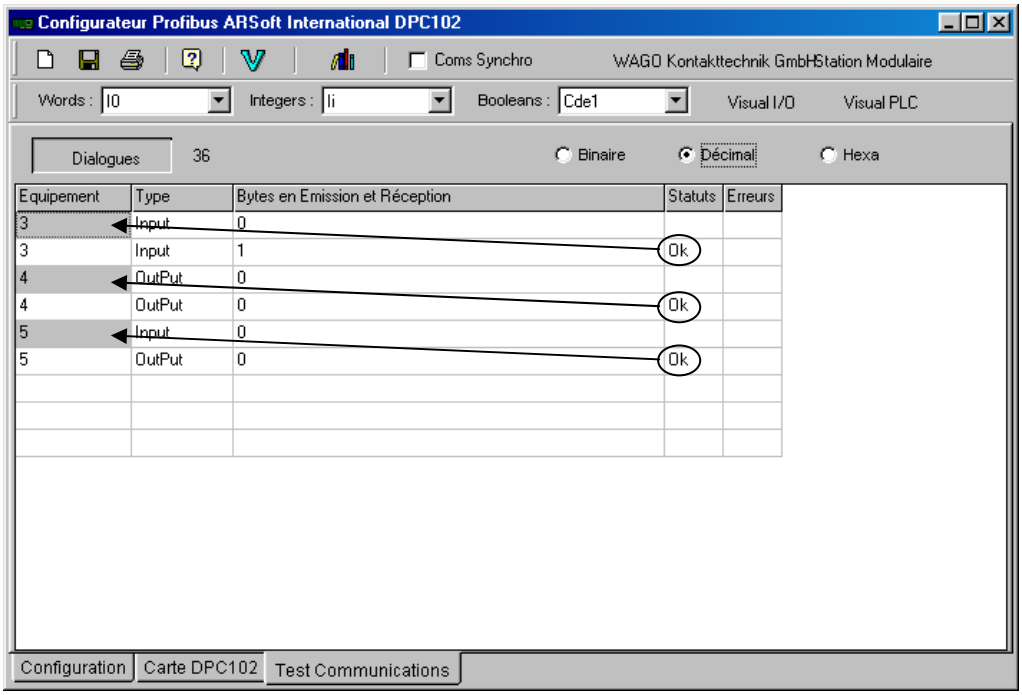
Cliquez sur le bouton **Test Carte** pour tester entièrement la carte Profibus (Mode « loop Back »)
Cette commande indique la position des Switches sur la carte si cette carte est au bus ISA ou PC104.



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 carte 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.

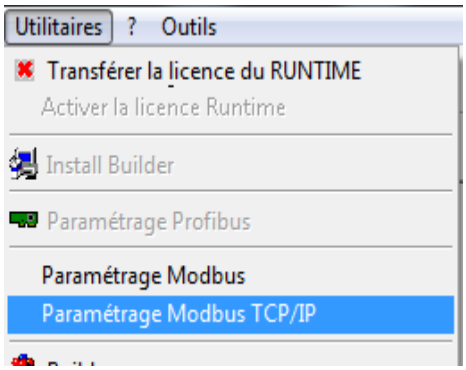
Fichiers nécessaires au fonctionnement de la carte Profibus.
DPConf.DP : Le fichier de configuration (le contenue de la grille) dans le répertoire de l'appli.
ARSBT.SYS: le driver Windows. Dans le répertoire Windows/driver
Arsdp.dll : La DLL d'interface avec le fichier de configuration dans le répertoire 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.

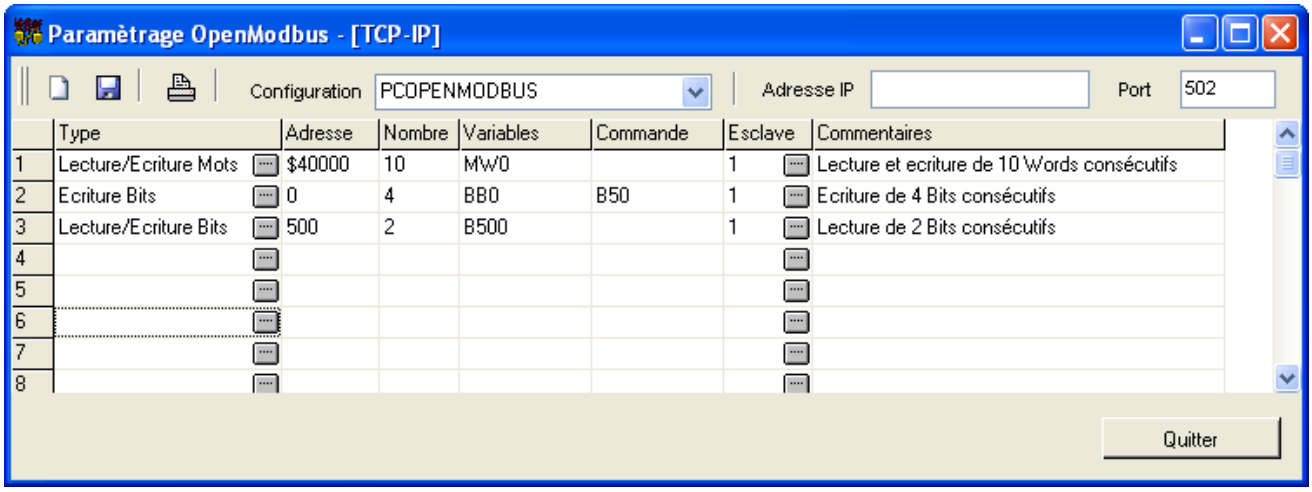
Coms synchro : Vous avez la possibilité de changer la fréquence d'activation du driver et de passer à 1 ms et multiples. Cependant considérant qu'un équipement Profibus réagit en 1 ms il est matériellement impossible de scanner à 1 ms plus d'un esclave Profibus.

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.

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 TCP. 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 commençant 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 PLC 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 commençant 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.

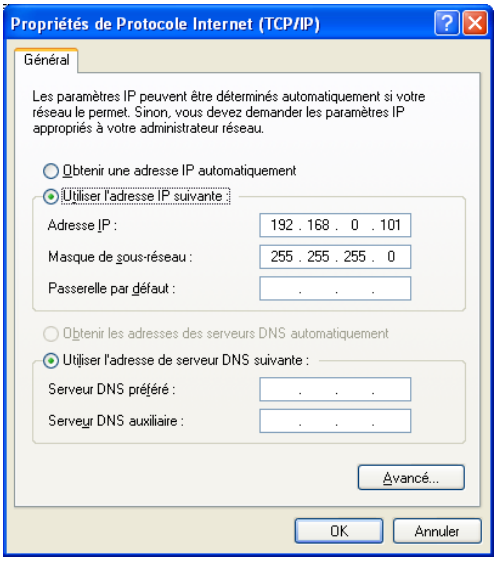
Le Driver OPENMODBUS (PCOpenModbus.VPU)
Après validation par le bouton Sauver, Visual PLC génère un driver compilé tenant compte des paramétrages. Ce driver est ensuite implémenté automatiquement dans l'application en cours (Menu : constitution application / taches prioritaires). Ce driver sera exécuté 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 (Câble 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..).
Note : Pour le Code 4 écrire 400000 (Adresse 00 000 to 99 999) Le Premier Chiffre indique le Code Modbus

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

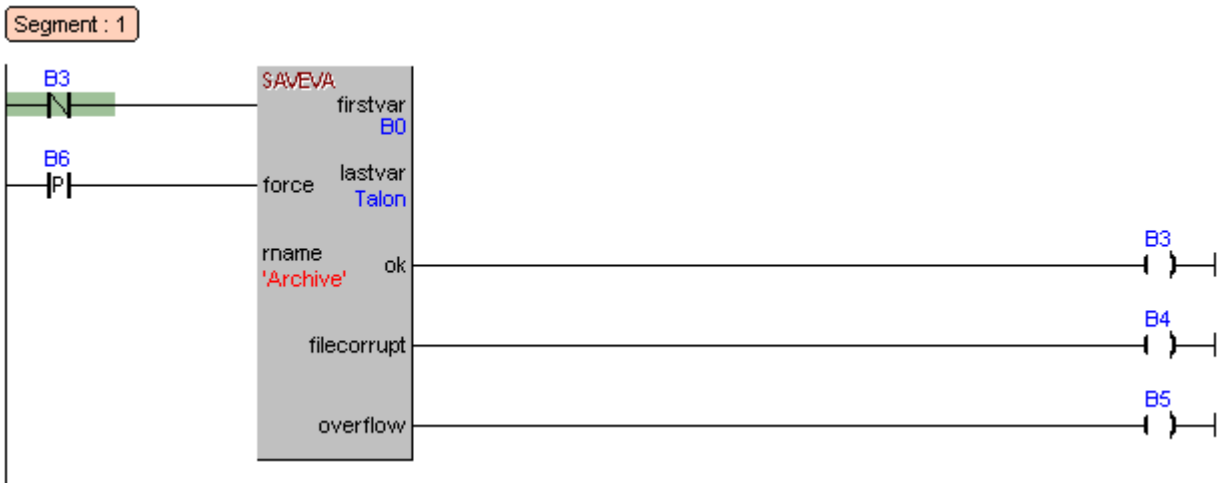
Types de communications possibles :

- 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

Note: Si vous devez lire ou écrire des variables flottantes dans l'automate, utilisez le type **Single** dans Visual IO et spécifiez 2 mots à lire ou écrire car 1 single = 4 octets = 2 mots.

Rémanence des données :

Est inclus dans Visual PLC le module fonctionnel SAVEVARS.MF.
Ce module fonctionnel compare les données et si celui-ci détecte un changement il stocke ces variables sur disque. Afin de limiter la taille du fichier SAVEVARS compresse les données.
Ajoutant aussi un calcul de CRC pour contrôler la validité de celles-ci au rechargement.



Ici B3 (Ok) est au repos au lancement de l'appli, validant ainsi l'auto chargement des variables globales.
Ensuite l'entrée force permet à souhait la sauvegarde en cours de fonctionnement.

Zone de surveillance des variables Globales :

Délimité par **FirstVar** et **Lastvar** (- 1 variable) ;

Ex de variables Globales :

```
  B0,B1,B2,B3,B4,B5,B6,B7,B8,B9              : Boolean;
  B10,B11,B12,B13,B14,B15,B16,B17,B18,B19    : Boolean;
  {******* 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.. **}
Talon : Boolean ;
```

Ici Zone sauvegardée va de B0 à MW29. (Talon est non compris).

Bit OK : Passe à True lors du démarrage de l'application rechargeant ainsi l'archive (Rname) dans les variables globales spécifiées par FirstVar et LastVar.

Bit File Corrupt : Passe à True lors du démarrage de l'application si le fichier archive est altéré. Dans ce cas aucun téléchargement n'est effectué.

Bit Overflow : Passe à True si la taille des variables globales (compressées) à sauvegarder est supérieur à 100Kbytes.

Force : Entrée permettant de forcer l'enregistrement des variables.
Cette entrée peut être utilisée après saisie de paramètres par l'opérateur puis validation par un bouton OK par exemple.
Cette entrée est sur front montant afin de sauvegarder une seule fois et non au tour de cycle.
Attention de ne pas mettre le bit Force (ici B6) dans la zone de sauvegarde, car au lancement de l'application ce bit repassera automatiquement à true.

Note :
Si l'entrée du module SaveVars est toujours valide, une comparaison cyclique des vars globales est effectuée et une sauvegarde si changement.

Ajouter un mot de passe pour l'accès à l'éditeur

Il est possible de verrouiller l'accès à l'éditeur en faisant apparaître au lancement de l'éditeur une boite de dialogue vous demandant de frapper un mot de passe que vous avez prédéfini.

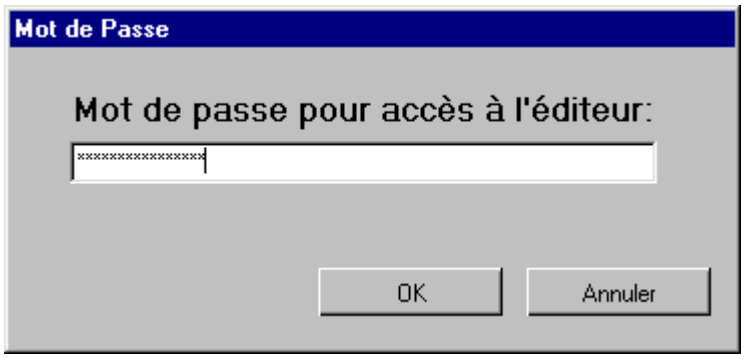
Ajoute un fichier texte (par exemple à l'aide de Notepad.exe) contenant sur une ligne le mot de passe que l'opérateur doit entrer au démarrage de l'éditeur.

Le fichier texte est **MP** (sans extension).

- Il doit être localisé dans la directorie d'installation (de base dans **C:\API32**)
- La première ligne contient le mot de passe à tester.

Exemple :
Première ligne = Mon Mot de Passe.

Au lancement cette fenêtre apparaît :



L'opérateur doit alors frapper **exactement** (attention aux majuscules/minuscules) le texte *Mon Mot de Passe,* pour accéder à l'éditeur.