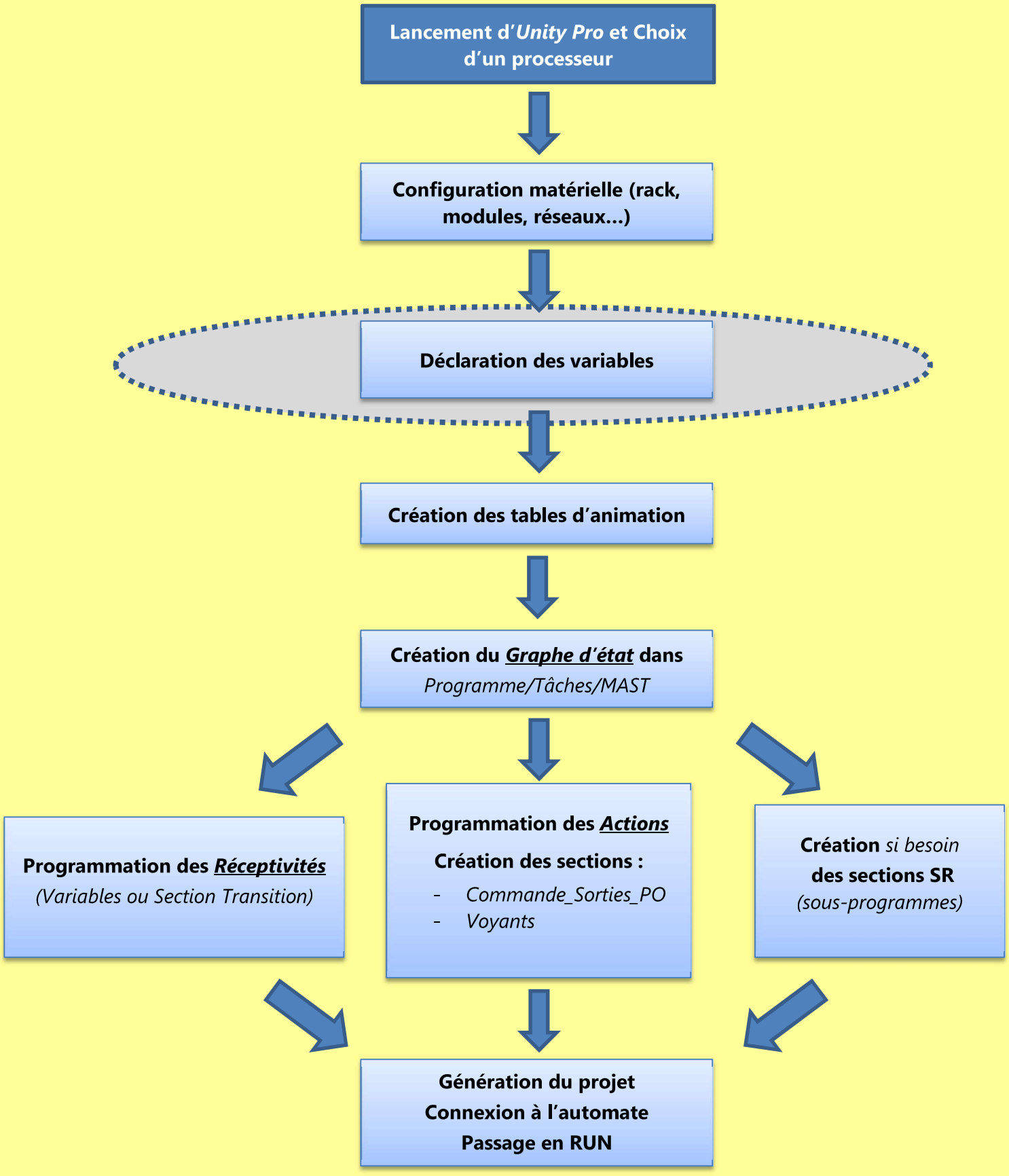
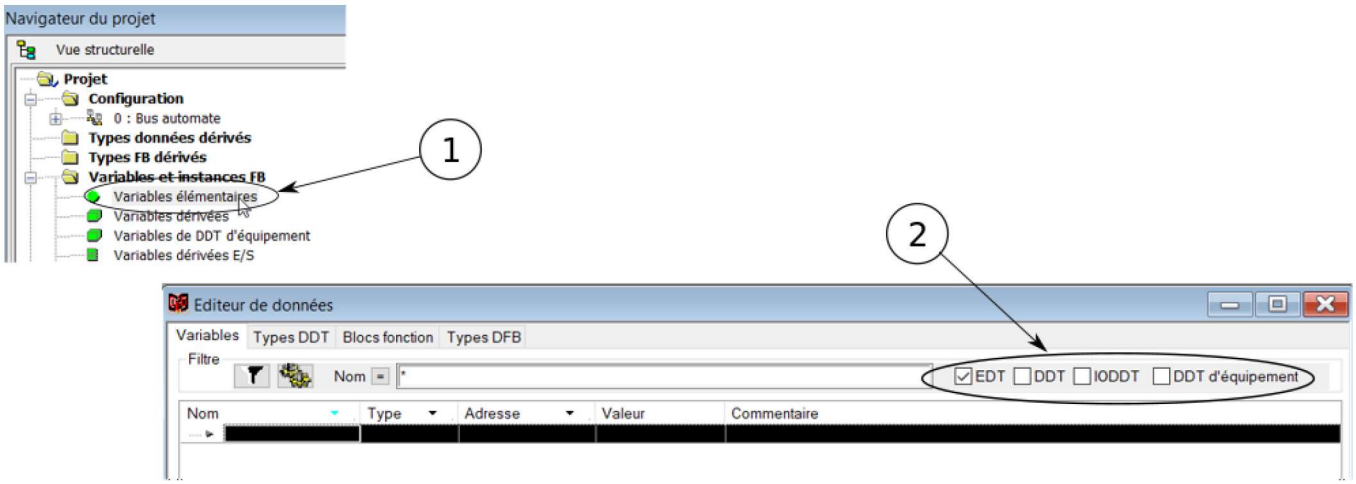
MÉTHODOLOGIE POUR LE DÉVELOPPEMENT D'UNE NOUVELLE APPLICATION



4 DÉFINITION DES VARIABLES

1. Généralités

- Une variable est une entité mémoire de type **BOOL**, **EBOOL**, **WORD**, **DWORD**, ... dont le contenu peut être modifié par le programme durant son exécution.
- Une **variable localisée** est une variable **liée à** un module d'entrées/sorties (ou module métier) associée à une zone mémoire spécifique (INT, DINT,...).
- Une **variable non localisée** est une variable **liée à aucun** module ou zone mémoire. Une variable qui n'a pas d'adresse est dite non localisée.
- Chaque variable doit être déclarée dans **l'éditeur de données**.
- Suivre les numéros entourés dans le schéma ci-dessous :
  1. **Clic²G** (ou **ClicD** puis, *Ouvrir*) sur *Variables élémentaires. On aurait pu accéder à l'éditeur de données, en* ***clic²G*** *sur l'un des 6 choix possibles (Variables élemtaires, Variables dérivées...) et même à partir du répertoire* ***Variables et instances FB***.
  2. Ici, la seule case cochée par défaut est **EDT**. Si on avait **clic²G** sur **Variables et instances FB**, toutes ces cases auraient été cochées.



L'**éditeur de données** contient toutes les variables de l'application *Unity*.

Quand on souhaite changer le nom (ou l'adresse, ou le commentaire) d'une variable, il faut **aller dans cet éditeur**.

On peut le renseigner **à chaque création de variable** ou bien lors de **l'utilisation d'une nouvelle variable** dans le programme automate.

Dans cet **éditeur**, on retrouve 4 onglets : *Variables*, *Types DDT*, *Blocs Fonction* et *Types DFB*.

2. Types de variables utilisées dans Unity

2.1 Données de type élémentaire EDT (Elementary Data Type)

- **BOOL** :
  Contient la valeur **0** ou **1** sans détection de fronts
- **EBOOL** :
  Contient la valeur **0** ou **1** avec détection de fronts
- **BYTE** :
  Chaîne de **8 bits**
- **INT** :
  Nombre entier de **16 bits signés** compris entre -32768 et $+32767$
- **UINT** :
  Nombre entier de **16 bits non signés**, valeur comprise entre 0 et 65535
- **DINT** :
  Nombre entier de **32 bits signés**, valeur comprise entre -2147483648 et $+2147483647$
- **UDINT** :
  Nombre entier de **32 bits non signés**, valeur comprise entre 0 et $+4294967295$
- **REAL** :
  Nombre réel codé sur **32 bits**, valeur comprise entre $-3.402824E+38$ et$-3.402824E+38$
- **WORD-DWORD** :
  Chaîne de **16 bits** ou **32 bits**. La particularité de ce format est que l'ensemble des bits qui le compose ne représente pas une valeur numérique, mais une combinaison de bits séparés. Très utilisé pour stocker des grandeurs de type BCD. Les données appartenant aux types de ce format peuvent être représentées sous 3 bases (l'hexadécimal, l'octal et le binaire).
- **TIME** :
  Codé sur **32 bits**. Contient **l'heure codée en millisecondes**, soit une durée maximale d'environ 49 jours.
  - La syntaxe est **T#** ou **TIME#** suivi de la valeur pouvant être exprimée en plusieurs unités.
    - T#4294967295mS (valeur en millisecondes)
    - T#4294967S_295MS (valeur en secondes et millisecondes)
    - T#4294967S_295MS (valeur en secondes et millisecondes)
    - T#71582M_47S_295MS (valeur en minutes, s et ms)
    - T#1193H_2M_47S_295MS (valeur en heures, min, s et ms)
    - T#49J_17H_2M_47S_295MS (valeur en jours, h, min, s et ms)
- **STRING** :
  Ce format permet de représenter une chaîne de caractères ASCII (lettre, chiffre, virgule...), chaque caractère étant codé sur un format de 8 bits. La taille maximale d'une chaîne est de 65534 caractères. La taille par défaut d'une variable ayant le format STRING est de 16 caractères.
  *Exemple : Soit la variable **toto** à qui on attribue un format **STRING**. toto :='il fait beau'.*
  Cette variable peut contenir **16 caractères**. Dans notre cas, elle en contient 12. On peut également définir le nombre de caractères que contiendra la variable STRING. Lors de l'affectation de l'attribut, il suffit de saisir **STRING[12]**.

2.2 Données complexes dérivées DDT (Derivated Data Type)

Les données de type dérivées (DDT) comprennent des données complexes de deux natures :

- **les tableaux**,
- **les structures** ;

Dans ce manuel, seules les **structures** seront abordées.

- **Structure** : C'est une donnée qui contient un ensemble de données de type différent telles que :
  - un ensemble de BOOL, WORD, UINT,... (structure EDT)
  - un ensemble de tableaux (structure DDT)
  - un ensemble de REAL, DWORD, tableaux (structure EDT et DDT)
- Les **structures** sont très utilisées dans les programmes d'automatismes pour créer des recettes. En effet, les machines de production peuvent en général fabriquer plusieurs produits avec des conditions de fabrication bien différentes. Chacune des recettes contiendra toutes les variables permettant la gestion respective d'un produit. Cela permet de **structurer** le programme.
  - Soit la fabrication de deux produits qui nécessitent chacun 10 variables pour assurer sa gestion. Au lieu de déclarer 20 variables de type EDT, on déclarera 2 variables de type Structures. Au sein de ces structures, on gèrera le type de variables que l'on a besoin.
  - Toutes les variables de type **EDT** et **DDT** sont à déclarer dans l'***Editeur de données***.

2.3 Données dérivées d'entrées sorties IODDT (Input Output Derivated Data Type)

Le terme **IODDT** désigne un type de données structuré représentant un module ou une voie d'un module automate. Chaque module expert présente ses propres **IODDT**. Les types **IODDT** sont prédéfinis par le constructeur. Ils contiennent des objets langage de la famille **EDT** appartenant à la voie d'un module métier. Ce sont des structures dont la taille (nombre d'éléments qui les composent) dépend de la voie ou du module d'entrées/sorties qu'elles représentent.

2.4 Données appartenant à un grafcet : la famille des types de données SFC

Cette famille regroupe des types de données dits composés tels que des structures restituant les propriétés et l'état du graphe **Chart** et des actions le composant.

Chaque **étape** est représentée par deux structures qui sont :

- la structure **SFCSTEP_STATE**
- la structure **SFCSTEP_TIMES**

- Analysons la structure **SFCSTEP_STATE** :
  - **x** : donnée élémentaire **EDT** de type **BOOL** contenant la valeur **TRUE** quand l'étape est **active**.
  - **t** : donnée élémentaire **EDT** de type **TIME** contenant le temps d'activité de l'étape (valeur en ms).
  - **tminErr** : donnée élémentaire **EDT** de type BOOL contenant la valeur **TRUE** si le temps d'activité de l'étape est inférieur au temps d'activité minimal programmé.
  - **tmaxErr** : donnée élémentaire **EDT** de type **BOOL** contenant la valeur **TRUE** si le temps d'activité de l'étape est supérieur au temps d'activité maximal programmé.

Syntaxe d'accès à des données de la structure **SFCSTEP_STATE** :

SYNTAXE	COMMENTAIRE
Nom_Etape.x	Permet de connaître l'état de l'étape (active/inactive)
Nom_Etape.t	Permet de connaître le temps d'activation en cours ou total de l'étape.
Nom_Etape.tminErr	Permet de connaître si le temps minimal d'activation de l'étape est **inférieur** au *temps de contrôle* dans **Nom_Etape.tmin**.
Nom_Etape.tmaxErr	Permet de connaître si le temps maximal d'activation de l'étape est **supérieur** au *temps de contrôle* dans **Nom_Etape.tmax**.

Se reporter à l'exemple de la page **??**.

- Analysons la structure **SFCSTEP_TIMES** :

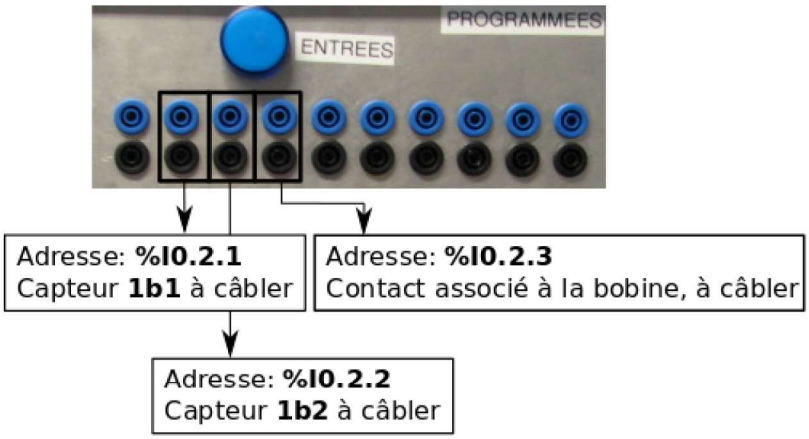
  Cette structure rassemble les données liées aux paramétrages du temps d'exécution de l'étape ou de la macro-étape.

  - **delay** : donnée élémentaire **EDT** de type **TIME** définissant le temps de retard de scrutation de la transition située en aval de l'étape active.
  - **tmin** : donnée élémentaire **EDT** de type **TIME** contenant la valeur minimale durant laquelle l'étape doit être exécutée. Si cette valeur n'est pas respectée, les données **tminErr** prennent la valeur **TRUE**.
  - **tmax** : donnée élémentaire **EDT** de type **TIME** contenant la valeur minimale durant laquelle l'étape doit être exécutée. Si cette valeur n'est pas respectée, les données tmaxErr prennent la valeur **TRUE**.

2.5 Choix de l'adressage des entrées de notre exemple : Machine à godets

Nous distinguons :

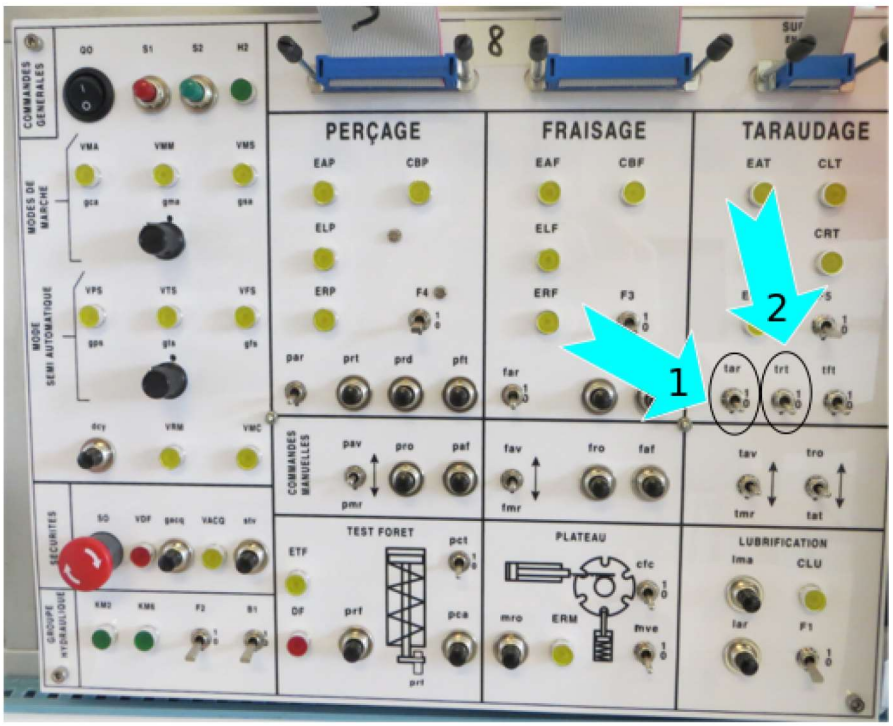
- **les entrées à câbler** (colonne **B** signifiant ***Banc***). Il s'agit :
  - des capteurs de fin de course du vérin **1b1** et **1b2**,
  - du contact associé à la bobine du contacteur du moteur **M1**.



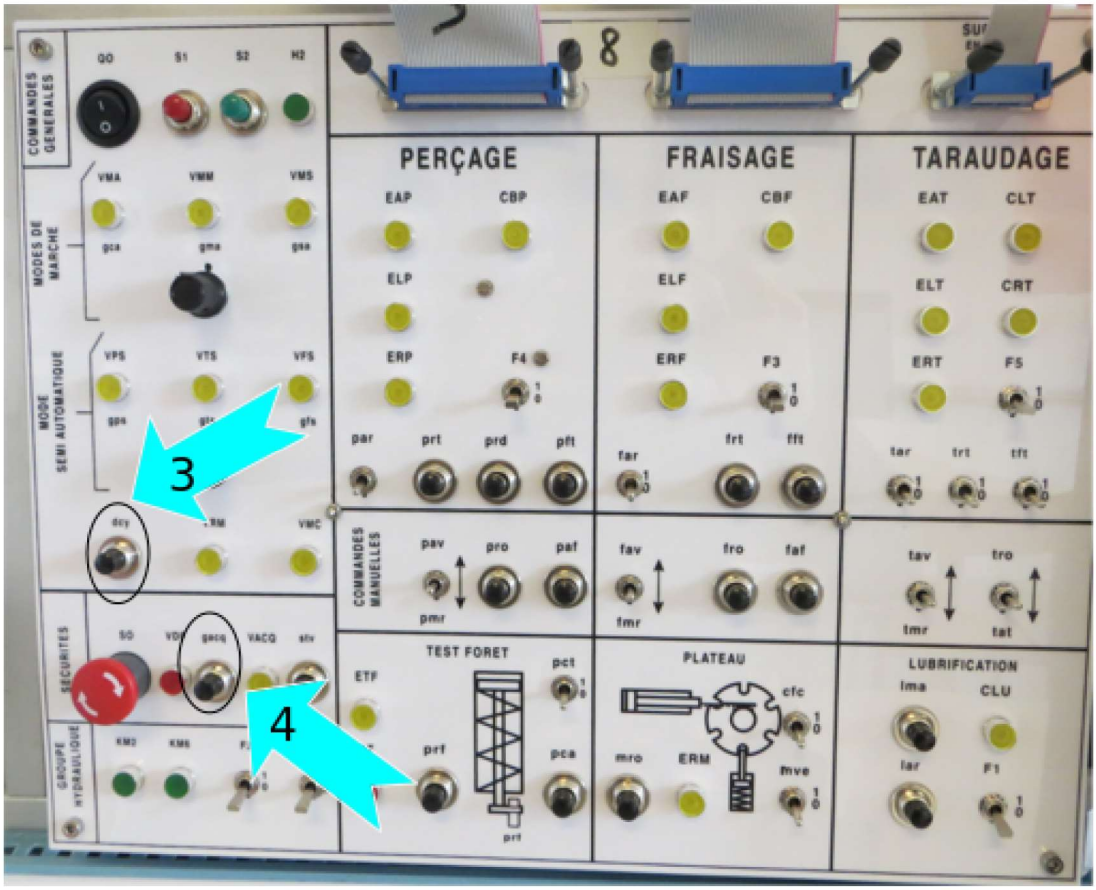
- **les entrées du pupitre déjà précâblées** (colonne **P** signifiant ***Pupitre***). Il s'agit :
  - du capteur indiquant la présence du godet sous la trémie **S1**,
  - du capteur indiquant la présence du godet en fin de convoyeur **S2**,
  - du bouton poussoir départ cycle **dcy**,
  - du bouton poussoir acquittement **acq**.

On n'a pas physiquement de capteurs indiquant la présence du godet sous la trémie et en fin de convoyeur. On va utiliser les interrupteurs *ci-dessous* situés sur le pupitre.

- Suivre les numéros entourés dans le schéma ci-dessous :
  1. On utilise **tar** pour le capteur de présence du godet *sous la trémie* (interrupteur vers le haut : S1 activé, présence godet *sous la trémie*)
  2. On utilise **trt** pour le capteur de présence du godet *en fin de convoyeur* (interrupteur vers le haut : S2 activé, présence godet *en fin de convoyeur*)



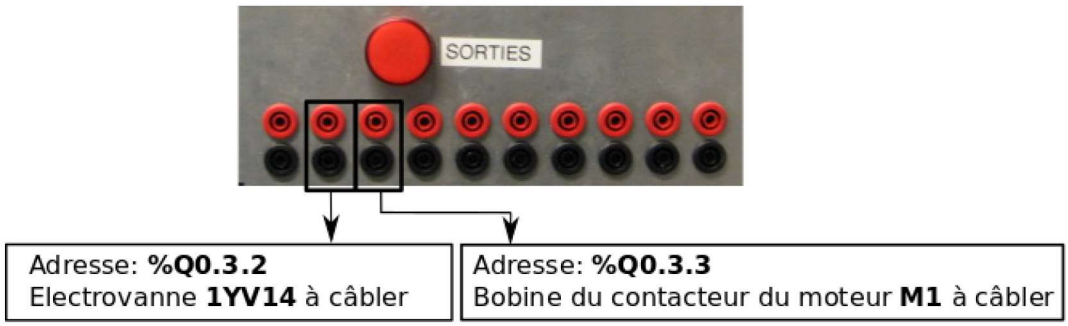
3. Le bouton poussoir nommé **dcy** servira pour le **départ cycle** (bouton poussoir appuyé : **dcy** activé)
4. Le bouton poussoir nommé **gacq** servira comme bouton poussoir d'acquittement **acq** (bouton poussoir appuyé : **acq** activé).



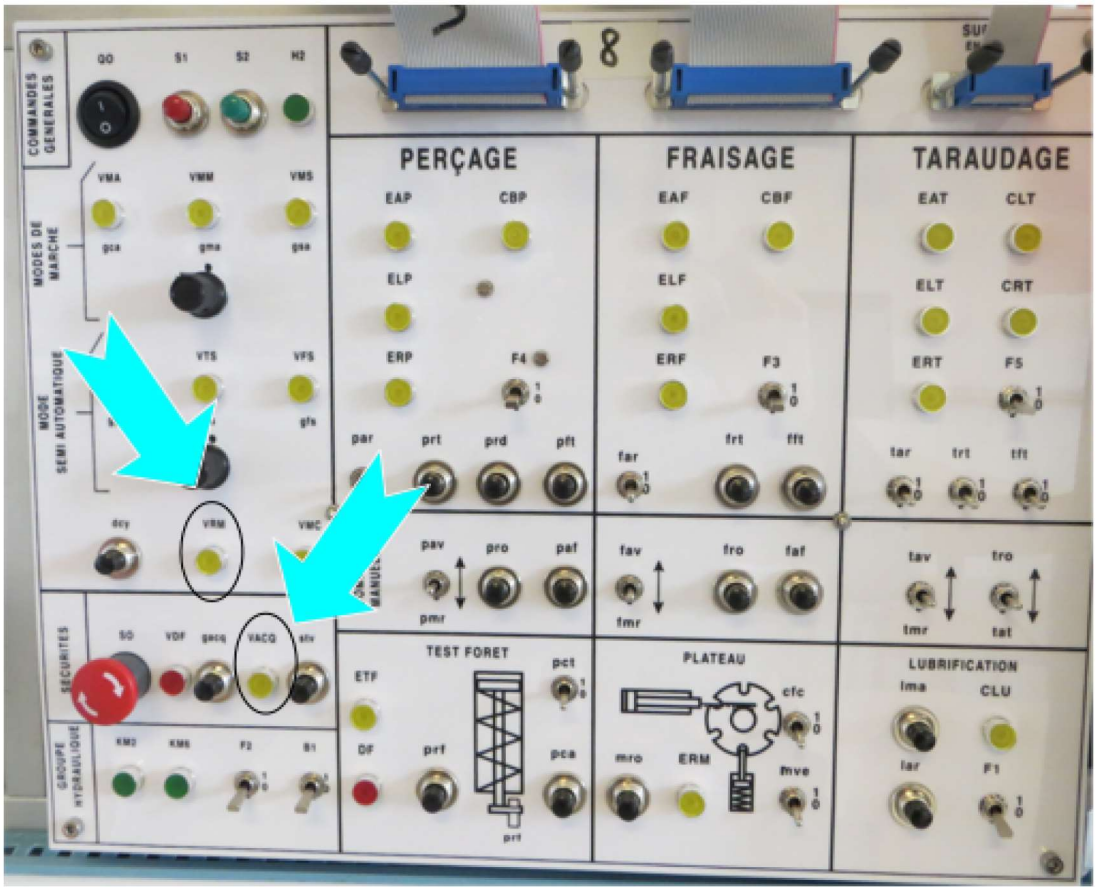
2.6 Choix de l'adressage des sorties de notre exemple : Machine à godets

Nous distinguons :

- **les sorties à câbler** (colonne **B** signifiant ***Banc***). Il s'agit :
  - de la commande de l'électrovanne **1YV14** qui permet de sortir le vérin,
  - de la commande de la bobine **KM1** qui permet de faire tourner le moteur.



- **les sorties du pupitre déjà précâblées** (colonne **P** signifiant ***Pupitre***). Il s'agit :
  - du voyant **VDCY**
  - du voyant **VACQ**



2.7 Tableau d'adressage de notre exemple : Machine à godets

⚠ Obligation de prendre cet adressage

ENTREES					
D*	**Explications**	**Symb* U***	**P***	**B***	**Ad***
1b1	Trappe fermée (vérin rentré)	_1b1		**x**	%I0.2.1
1b2	Trappe ouverte (vérin sorti)	_1b2		**x**	%I0.2.2
km1	Contacts du contacteur du moteur M1	ckm1		**x**	%I0.2.3
S1	Présence godet sous trémie	S1	**x** *tar*		%I0.2.11
S2	Présence godet en fin de convoyeur	S2	**x** *trt*		%I0.2.12
dcy	Bouton poussoir départ cycle	dcy	**x** *dcy*		%I0.2.32
acq	Bouton poussoir acquittement	acq	**x** *gacq*		%I0.2.33

SORTIES					
D*	**Explications**	**Symb* U***	**P***	**B***	**Ad***
1YV14	OUVRIR la trappe (ev*)	_1YV14		**x**	%Q0.3.2
KM1	AVANCER le tapis (bobine du contacteur)	KM1		**x**	%Q0.3.3
VDCY*	Voyant départ cycle	VDCY	**x** *VRM*		%Q0.3.5
VACQ*	Voyant d'acquittement	VACQ	**x** *VACQ*		%Q0.3.25

Légende :

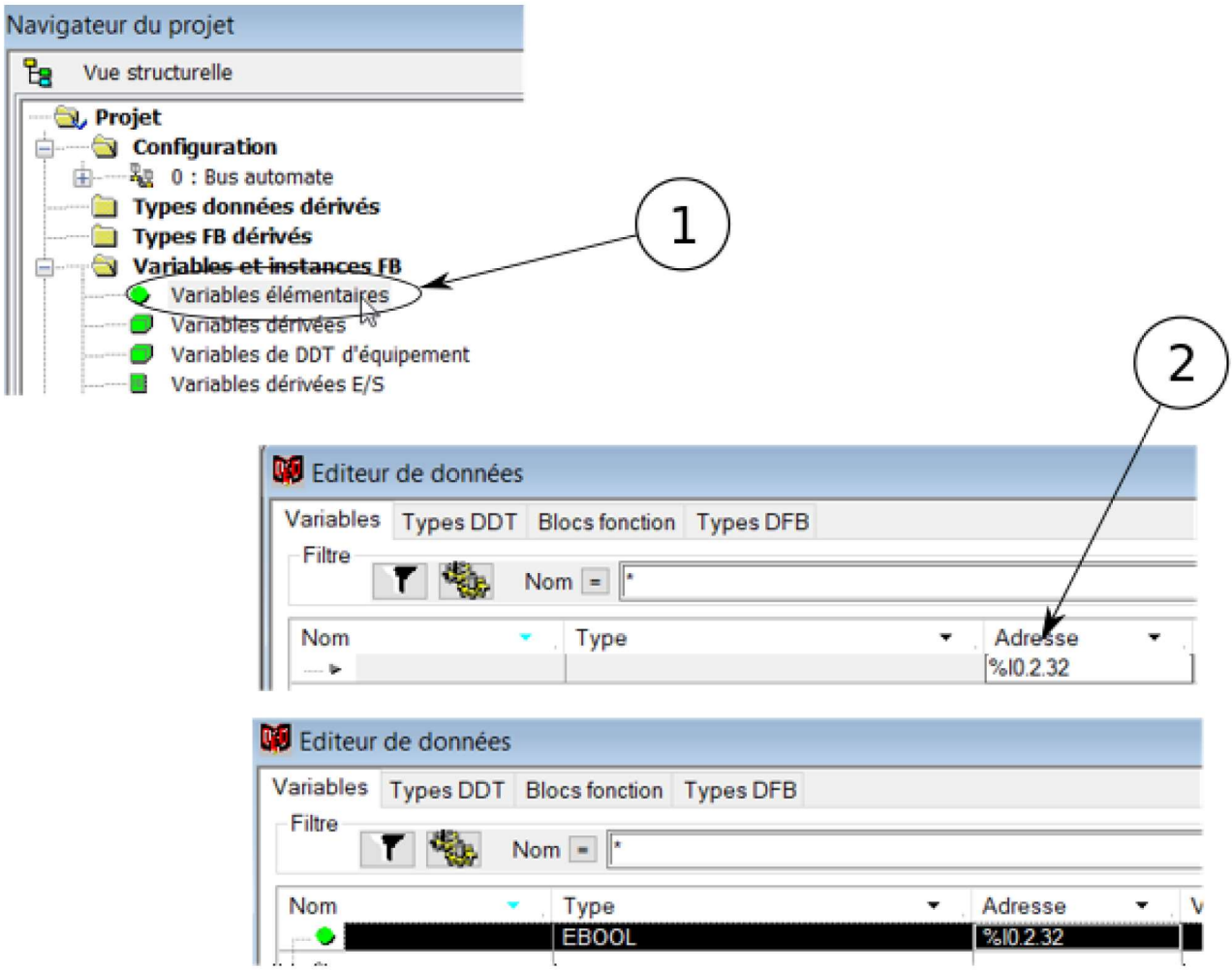
- Symb* U* : symbole Unity
- P* : pupitre
- B* : banc
- Ad* : adresse automate
- Ev* : électrovanne
- * : Clignotant toutes les 0.5s

3. Création de variables sous l'éditeur de données

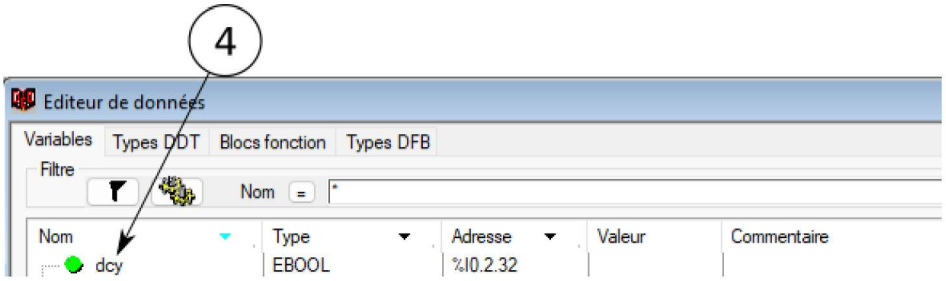
3.1 Création d'une variable localisée de type EBOOL

- Suivre les numéros entourés dans le schéma ci-dessous :

1. **Clic²G** sur *Variables élémentaires*.
2. Taper au clavier «%**I**0.2.32» ; **I** pour **I**nput, puis valider.
3. Le type «**EBOOL**» de la variable **dcy** apparaît.

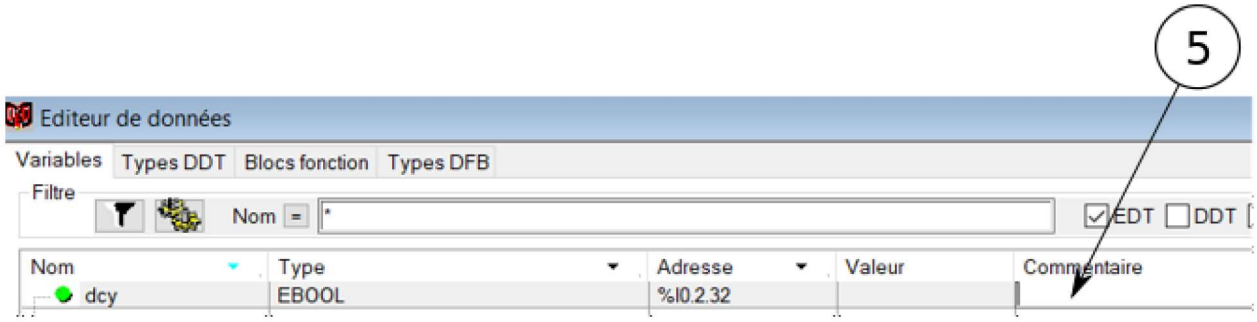


4. **Clic²G** dans le champ repéré par le numéro **4**, et taper **dcy_api**.



En suivant cette procédure, la variable **dcy** localisée de type «EBOOL» a été bien créée.

5. **Clic²G** pour taper le commentaire : «**Bouton poussoir départ cycle**».



Il est important d'écrire des commentaires afin de faciliter la lecture du programme

6. Créer toutes les variables dans l'éditeur de la «machine à godets».

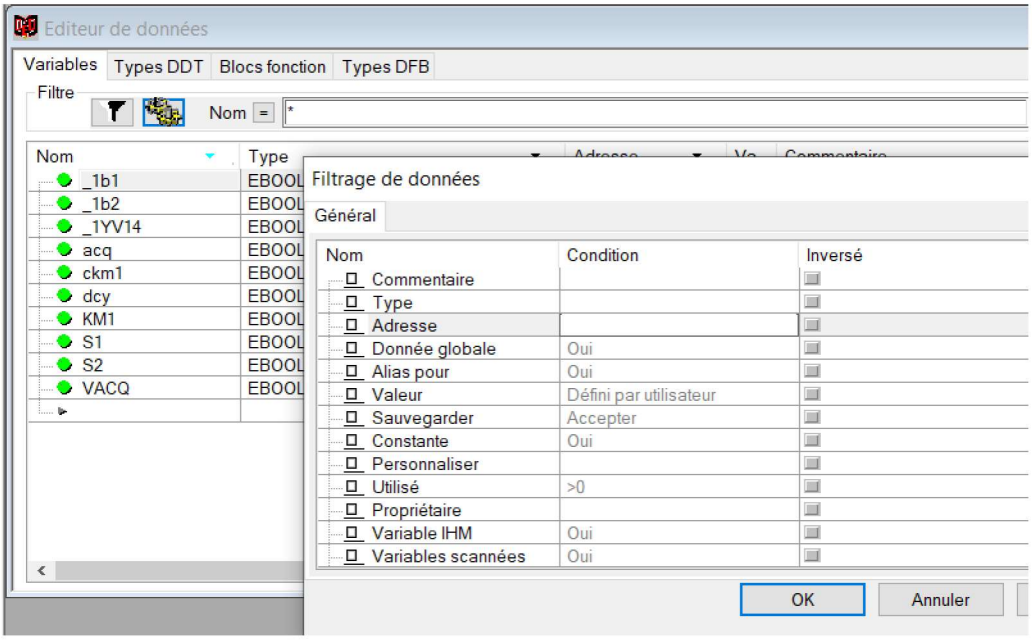
3.2 Editeur à obtenir pour l'exercice MACHINE À GODETS

Voici l'éditeur que vous devez obtenir :

Nom	Type	Adresse	Valeur	Commentaire	Horodatage	Droits lecture/écriture (R/W) de la vari
_1b1	EBOOL	%I0.2.1		Trappe fermée	Aucun	
_1b2	EBOOL	%I0.2.2		Trappe ouverte	Aucun	
ckm1	EBOOL	%I0.2.3		Contact du contacteur	Aucun	
S1	EBOOL	%I0.2.11		Présence godet sous trémie	Aucun	
S2	EBOOL	%I0.2.12		Présence godet en fin de convoyeur	Aucun	
dcy	EBOOL	%I0.2.32		Bouton poussoir départ cycle	Aucun	
acq	EBOOL	%I0.2.33		Bouton poussoir acquittement	Aucun	
_1YV14	EBOOL	%Q0.3.2		OUVRIR la trappe	Aucun	
KM1	EBOOL	%Q0.3.4		AVANCER le tapis (bobine du contacteur)	Aucun	
VDCY	EBOOL	%Q0.3.5		VOYANT départ cycle	Aucun	
VACQ	EBOOL	%Q0.3.25		VOYANT d'acquittement	Aucun	

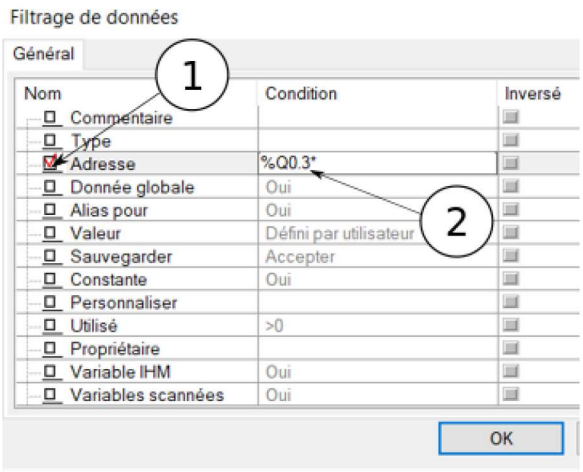
3.3 Filtrage de données dans l'éditeur

Le filtrage des données est intéressant quand un grand nombre de variables sont présentes dans **l'éditeur de données**. On peut alors **via les options de filtre (2 roues dentées)** ne faire afficher que celles désirées.

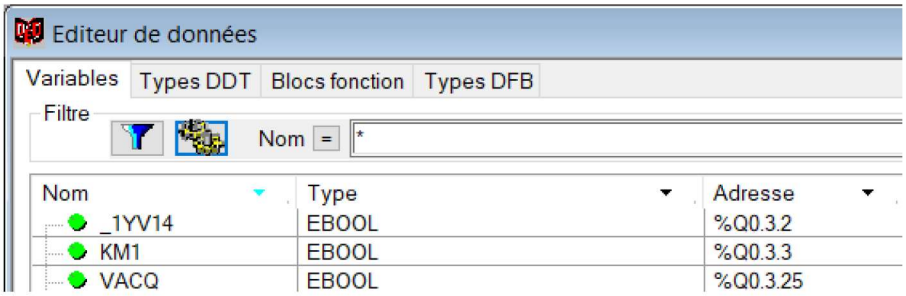


Exemple : Visualisation uniquement des sorties.

1. On coche ***Adresse***.
2. Dans ***Condition***, on écrit **%Q0.3***. Ainsi, on demande l'affichage de toutes les sorties commençant par **%Q0.3** (le symbole * permet de spécifier toutes).

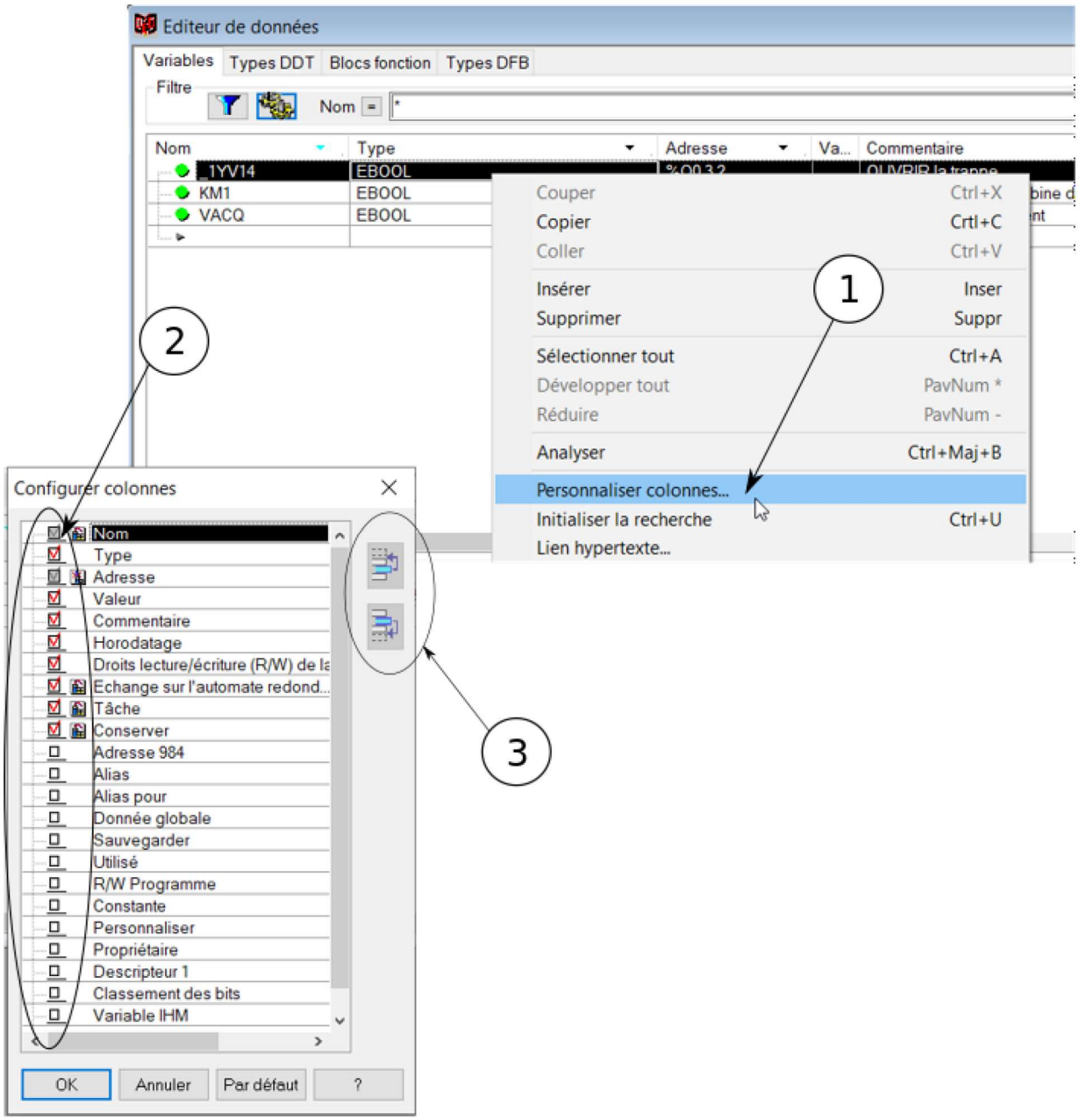


La fenêtre obtenue est :



Il n'y a bien évidement que des **sorties affichées**.

1. **ClicD** dans l'**éditeur de données**, ainsi une fenêtre apparaît. On choisit ***Personnaliser colonnes***
2. Vous pouvez choisir les éléments à afficher
3. Vous pouvez modifier la position des colonnes via ces icônes.


Editeur de données
Variables
Types DDT
Blocs fonction
Types DFB
Filtre
Nom
Type
Adresse
Commentaire
_1YV14
EBOOL
KM1
EBOOL
VACQ
EBOOL
Couper Ctrl+X
Copier Crtl+C
Coller Ctrl+V
Insérer Inser
Supprimer Suppr
Sélectionner tout Ctrl+A
Développer tout PavNum *
Réduire PavNum -
Analyser Ctrl+Maj+B
Personnaliser colonnes...
Initialiser la recherche Ctrl+U
Lien hypertexte...
1
2
3
Configurer colonnes
Nom
Type
Adresse
Valeur
Commentaire
Horodatage
Droits lecture/écriture (R/W) de la
Echange sur l'automate redond...
Tâche
Conserver
Adresse 984
Alias
Alias pour
Donnée globale
Sauvegarder
Utilisé
R/W Programme
Constante
Personnaliser
Propriétaire
Descripteur 1
Classement des bits
Variable IHM
OK
Annuler
Par défaut
?