Notion de blocs

Objectif

Dans ce TP, vous allez vous familiariser avec les différents blocs utilisés dans la programmation du SIMATIC S7-1200 avec le logiciel TIA Portal. Ce TP explique les différents types de bloc et montre à travers les étapes ci-dessous comment créer un programme dans un bloc de fonction.

- Création du bloc fonction
- Définition des variables locales
- Programmation avec les variables locales dans le bloc fonction
- Appel et paramétrage du bloc fonction dans OB1

Pré-requis

Les connaissances suivantes sont requises pour l'étude de ce module :

- Notions de base en programmation d'API avec TIA Portal (TP1– « Initiation à la programmation du SIMATIC S7-1200 avec TIA Portal VX »)

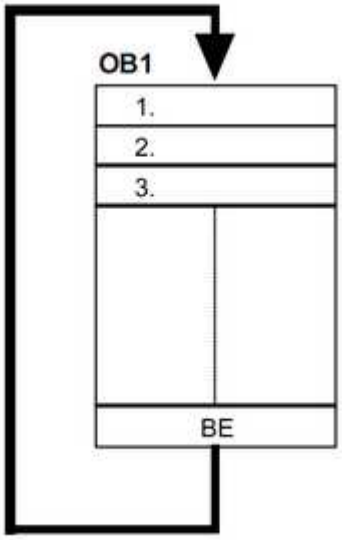
1 Types de blocs pour le SIMATIC S7-1200

Pour le SIMATIC S7-1200, le programme est écrit dans ce qu'on appelle des **blocs**.
De base, un bloc d'organisation **OB1** est créé lors de l'ajout d'une CPU.
Ce bloc représente l'interface du système d'exploitation de la CPU. Il est appelé automatiquement par celle-ci, et est traité de manière cyclique.

Lors du traitement du programme, il y a deux possibilités différentes, dépendantes de l'appareil de commande utilisé et de la programmation.
Le traitement de chaque instruction nécessite un certain temps (de l'ordre de la microseconde). On appelle durée de cycle, la durée d'un (seul) traitement de toutes les instructions. C'est le temps d'un cycle programme.

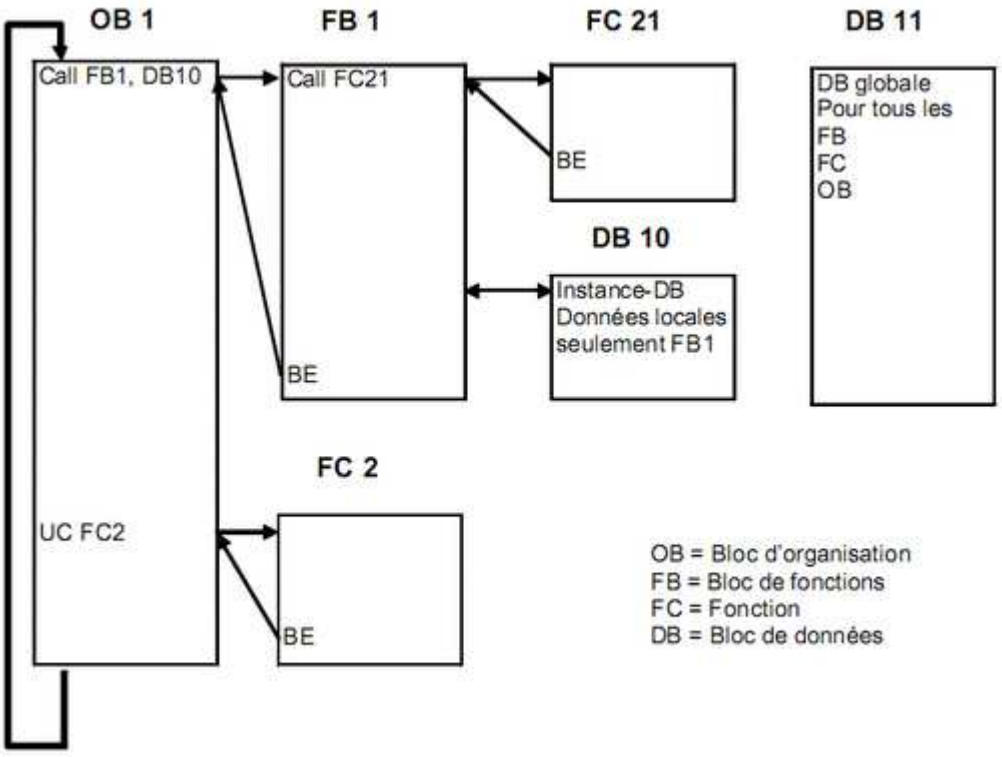
1.1 Traitement linéaire du programme

Lors de la programmation linéaire, les instructions de l'appareil de commande sont traitées les unes après les autres, telles qu'écrites dans la mémoire programme. Si la fin du programme (BE) est atteinte, le traitement du programme recommence du début.
On parle de traitement cyclique.
La durée, nécessaire à un appareil pour une itération de traitement, est appelée durée de cycle. Le traitement linéaire du programme est la plupart du temps utilisé pour des commandes simples et pas trop volumineuses. Il peut être réalisé dans un seul OB.



1.2 Traitement structuré du programme

On répartit le programme d'un ensemble volumineux de tâches de commande en blocs de programmes petits, clairs, associés à des fonctions. Cela présente l'avantage de pouvoir tester les blocs de manière individuelle et de les faire fonctionner ensemble par une fonction globale.
Les blocs de programme doivent être appelés par des commandes d'appel de blocs (Call xx / UC xx / CC xx). Si la fin du bloc est reconnue, le programme qui a appelé le bloc est de nouveau traité.



1.3 Blocs utilisateur pour le SIMATIC S7-1200

STEP 7 offre pour la programmation structurée les blocs utilisateur suivants :

- **OB (Bloc Organisation) :**
  Un OB est appelé cycliquement par le système d'exploitation et réalise ainsi l'interface entre le programme utilisateur et le système d'exploitation. Le dispositif de commande est informé dans cet OB par des commandes d'appel de blocs, de quels blocs de programme il doit traiter.
- **FB (Bloc de fonction) :**
  Le FB est à disposition via un espace mémoire correspondant. Si un FB est appelé, il lui est attribué un bloc de données (DB). On peut accéder aux données de cette instance DB par des appels depuis le FB. Un FB peut être attribué à différents DB. D'autres FB et d'autres FC peuvent être appelés dans un bloc de fonction par des commandes d'appel de blocs.
- **FC (Fonction) :**
  Une FC ne possède pas un espace mémoire attribué. Les données locales d'une fonction sont perdues après le traitement de la fonction. D'autres FB et FC peuvent être appelés dans une fonction par des commandes d'appel de blocs.
- **DB (Bloc de données) :**
  Les DB sont employés afin de tenir à disposition de l'espace mémoire pour les variables de données. Il y a deux catégories de blocs de données. Les DB globaux où tous les OB, FB et FC peuvent lire des données enregistrées et écrire eux-mêmes des données dans le DB. Les instances DB sont attribuées à un FB défini.

Indication :

Si pendant la programmation des FC et des FB seules les variables locales sont utilisées, alors ces blocs peuvent être utilisés et appelés une multitude de fois sous la forme de blocs standards.
Toutefois, les blocs FB doivent avoir une zone mémoire attribuée, une instance (comme par exemple un DB), à chaque fois qu'ils sont appelés.

1. Blocs d'organisation

Les blocs d'organisation (OB) sont l'interface entre le système d'exploitation et le programme utilisateur.
Ils sont appelés par le système d'exploitation et gère les opérations suivantes :

- Comportement au démarrage de l'automate
- Traitement cyclique du programme
- Traitement par programme des alarmes de processus
- Gestion des erreurs

Vous pouvez programmer les blocs d'organisation comme vous le voulez, et donc déterminer le comportement de la CPU.

Vous avez plusieurs options d'utilisation des blocs d'organisation dans votre programme :

- **OB de démarrage** (*Startup OB*), **OB cycliques** (*Program Cycle OB*), **OB d'erreur de temps** (*Time Error Interrupt OB*), et **OB d'alarme de diagnostic** (*Diagnostic Error OB*) :

  Vous pouvez insérer et programmer simplement ces blocs d'organisation dans votre projet. Vous n'avez ni besoin de les régler, ni de les appeler.

- **OB d'alarme de processus** (*Hardware Interrupt OB*) et **OB d'alarme cyclique** (*Cyclic Interrupt OB*) :

  Ces blocs d'organisation doivent être paramétrés après leur insertion dans le programme. De plus, les OB d'alarme de processus peuvent être affectés à un évènement au moment de leur exécution en utilisant l'instruction ATTACH, ou séparés grâce à l'instruction DETACH.

- **OB d'alarme temporisée** (*Time Delay Interrupt OB*) :

  Les OB d'alarme temporisée peuvent être insérés dans votre projet et programmés. De plus, vous devez les appeler dans le programme utilisateur grâce à l'instruction SRT_DINT. Un paramétrage n'est pas nécessaire.

Informations au niveau du démarrage des OB
Quand les quelques blocs d'organisation sont démarrés, le système d'exploitation parcourt les informations qui peuvent être évaluées dans le programme utilisateur.
Ceci peut être très utile pour les diagnostics d'erreur.
Les descriptions des blocs d'organisation indiquent si une information a bien été lue, et laquelle.

2. Fonctions

Une fonction contient un programme qui est exécuté quand un autre bloc de code appelle cette fonction.

Les fonctions (FC) sont des blocs de code sans mémoire. Les données des variables temporaires sont perdues après que la fonction a été traitée. Les blocs de données globaux peuvent être utilisés pour stocker les données des FC.

Les fonctions peuvent être utilisées pour les cas suivants, par exemple :

- Retourner des valeurs de fonction au bloc d'appel, par exemple dans le cas des fonctions mathématiques ;
- Exécuter des fonctions technologiques, par exemple des commandes individuelles avec des opérations binaires.

Une fonction peut également être appelée plusieurs fois à divers endroits du programme. Ceci facilite la programmation de fonctions complexes et répétitives.

3. Blocs de fonction

Les blocs de fonction contiennent des sous-programmes qui sont toujours exécutés quand un bloc de fonction est appelé par un autre bloc de code.

Les blocs de fonction sont des blocs de code qui stockent leurs valeurs dans des instances DB, ceci afin que ces valeurs soient disponibles même après que le bloc a été traité.
Stocker vos paramètres d'entrées, de sorties et d'entrées/sorties dans des instances DB rend ces paramètres accessibles en permanence, après que le bloc a été traité : pour cette raison, ils sont aussi appelés blocs avec « mémoire ».

Les FB sont utilisés pour des tâches qui ne peuvent être mis en œuvre avec des fonctions :

- Toujours quand les tempos et les compteurs sont nécessaires dans un bloc.
- Toujours quand les informations doivent être stockées dans le programme ; par exemple, quand on présélectionne un mode de fonctionnement avec un bouton.

Un bloc de fonction peut aussi être appelé plusieurs fois à divers endroits du programme. Ceci facilite la programmation de fonctions complexes et répétitives.

Instances des FB :
Le fait d'appeler un FB est nommé une instance.
Pour chaque instance d'un FB, une zone mémoire lui est affectée, contenant les données utiles au traitement du bloc. Cette mémoire est fournie par des blocs de données que le logiciel génère automatiquement. Il est également possible de fournir de la mémoire pour plusieurs instances grâce un bloc de données en **multi-instance**.

4. Blocs de données

Contrairement aux blocs de code, les blocs de données ne contiennent pas d'instructions, mais ils sont utilisés pour stocker les données utilisateur. Ceci signifie que les blocs de données contiennent des données variables dont le programme utilisateur se sert pour le traitement du programme.

Les **blocs de données globaux** stockent des données qui peuvent être utilisés par tous les autres blocs.
La taille maximale des DB varie selon la CPU. La structure des blocs de données globaux peut être définie comme on le souhaite.

Quelques exemples d'application :

- Stockage d'informations pour la gestion d'un magasin : « Tel produit se situe ici ».
- Stockage d'informations sur certains produits.

Tous les FB, FC ou OB peuvent lire ou écrire les données d'un bloc de données global. Ces données sont conservées dans le DB, même quand il est quitté.

L'appel d'un bloc fonction est appelé une instance. Pour chaque appel d'un FB avec transfert de paramètres, un **DB d'instance** lui est affecté et sert de stockage de données. Ainsi, les paramètres locaux et les données statiques des FB sont stockés dedans.
La taille maximale des DB d'instance varie également selon la CPU. Les variables déclarées dans le FB déterminent la structure du DB d'instance.

Un bloc de données global et un bloc de données d'instance peuvent être ouverts en même temps.

2 Exemple d'application : Commande d'un convoyeur

Quand les blocs sont créés, s'ils doivent travailler dans un programme quelconque qu'on pourrait appeler « boîte noire », ils doivent être programmés en utilisant des variables. Dans ce cas, la règle suivante s'applique : dans ces blocs, seules les entrées/sorties à adresse non-absolue, les mnémoniques, etc… doivent être utilisées. Dans ces blocs, seules les variables et les constantes sont utilisées.

Dans l'exemple ci-dessous, un bloc de fonction doit être créé avec une déclaration de variable qui contient une commande pour un convoyeur dépendante du mode de fonctionnement choisi.

Avec le bouton « S1 », on peut sélectionner le mode de fonctionnement « Manuel », et avec le bouton « S2 », on peut sélectionner le mode « Automatique ».
En mode « Manuel », le moteur est alimenté tant qu'on appuie sur le bouton « S3 » et que le bouton « S4 » n'est pas activé.
En mode « Automatique », le moteur du convoyeur est allumé avec le bouton « S3 » et éteint avec le bouton « S4 ».

Tableau d'affectations

Adresses	Variables	Commentaires
%I 0.0	S1	Bouton mode manuel, S1 (NO)
%I 0.1	S2	Bouton mode automatique, S2 (NO)
%I 0.2	S3	Bouton marche, S3 (NO)
%I 0.3	S4	Bouton arrêt, S4 (NF)
%Q 0.2	M01	Moteur du convoyeur M01

Indication
Le bouton arrêt S4 est un contact NF pour une raison de sécurité en cas de rupture de fil. En effet, si le fil de ce bouton casse, le système s'arrête automatiquement. Autrement, le système ne pourrait pas être stoppé si ce fil se cassait. C'est pour cette raison qu'en automatique tous les boutons ou commutateurs stop/arrêt sont des contacts à ouverture.

3 Programmation de la commande d'un convoyeur pour le SIMATIC S7-1200

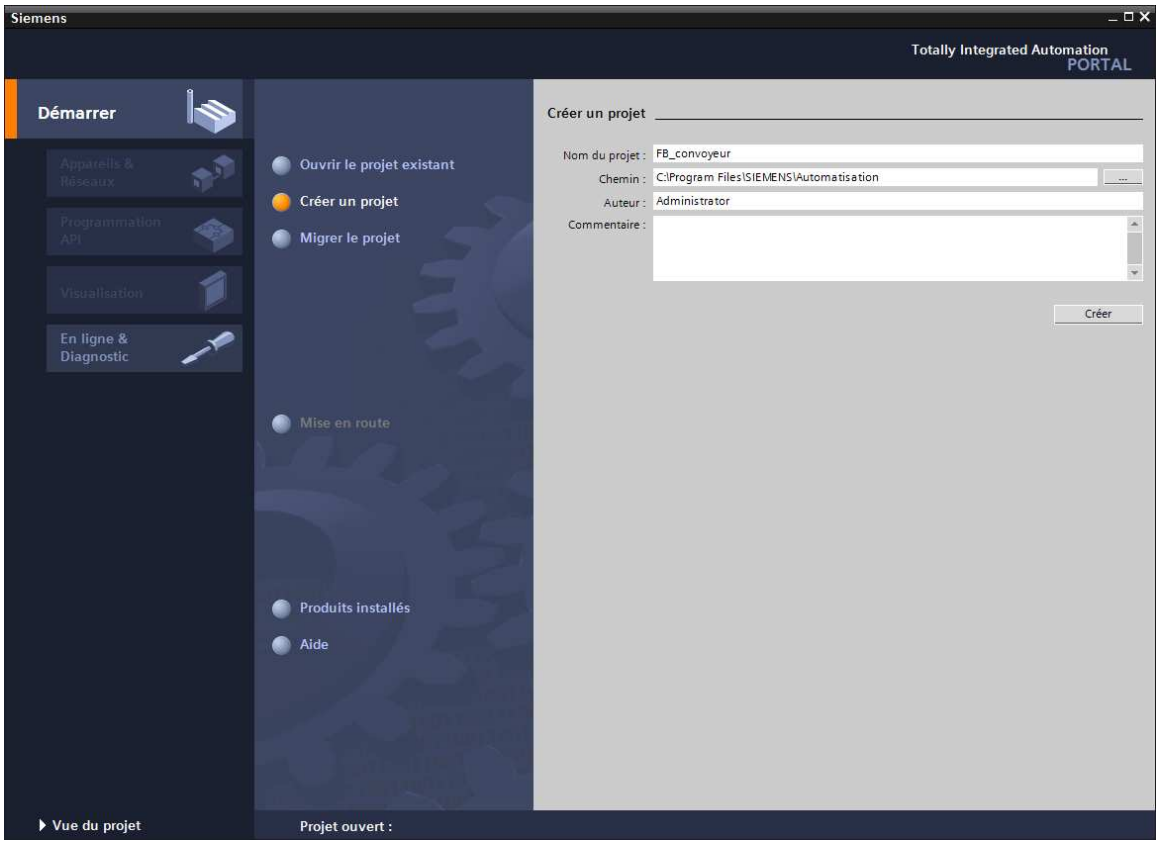
La gestion du projet et sa programmation se font grâce au logiciel « **Totally Integrated Automation Portal** ».

Là, sous une même interface, les éléments tels que le système de contrôle, la visualisation et la mise en réseau de la solution d'automatisation sont créés, paramétrés et programmés.
Les outils en ligne sont disponibles pour les diagnostics d'erreur.

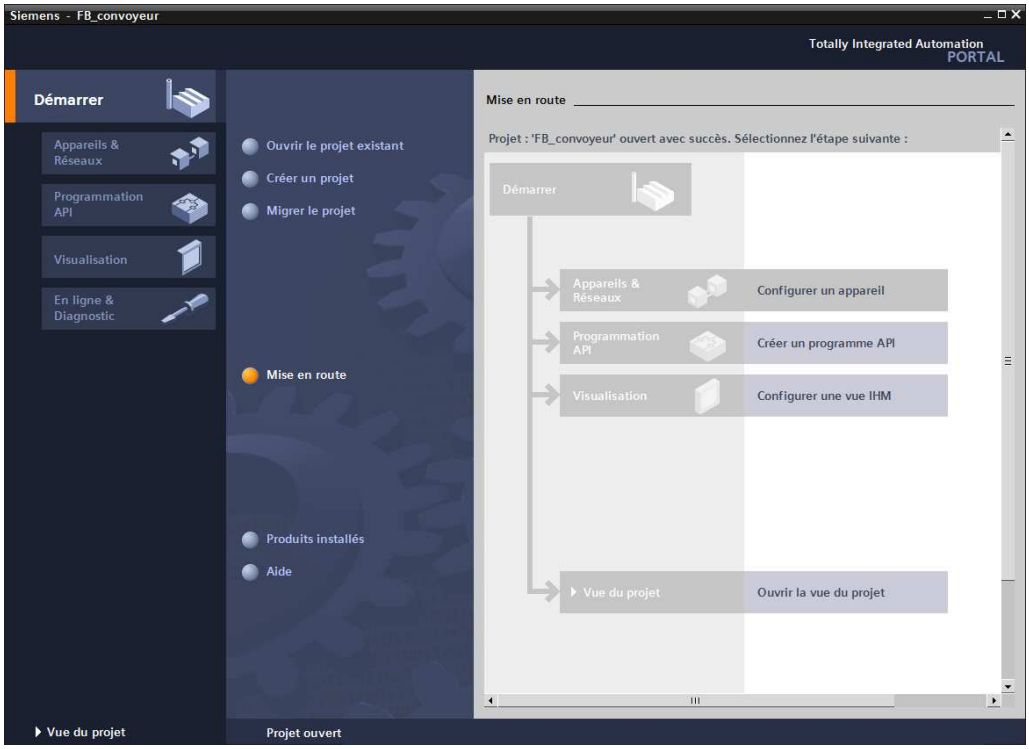
Les étapes ci-dessous montrent comment créer un projet pour SIMATIC S7-1200 et programmer la solution pour cette application.

1. L'outil que nous allons utiliser est « **Totally Integrated Automation Portal** », que l'on appelle ici d'un double-clic.

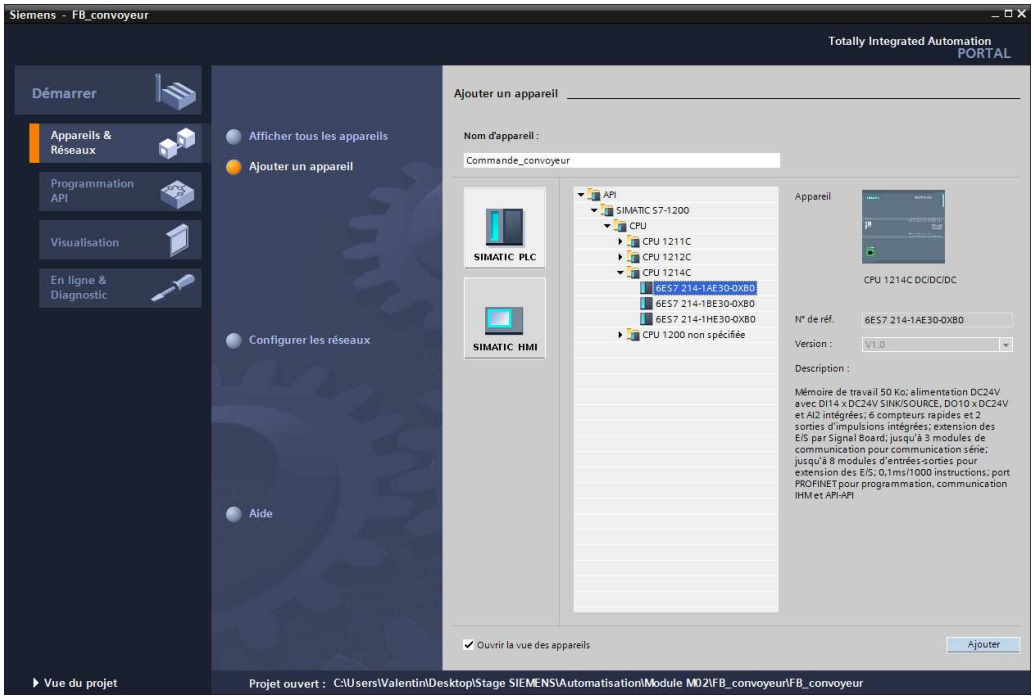
2. Les programmes pour SIMATIC S7-1200 sont gérés sous forme de projets. Nous allons maintenant créer un nouveau projet via la vue portail (« **Créer un projet** > **Nom** : *FB_convoyeur* > **Créer** »).



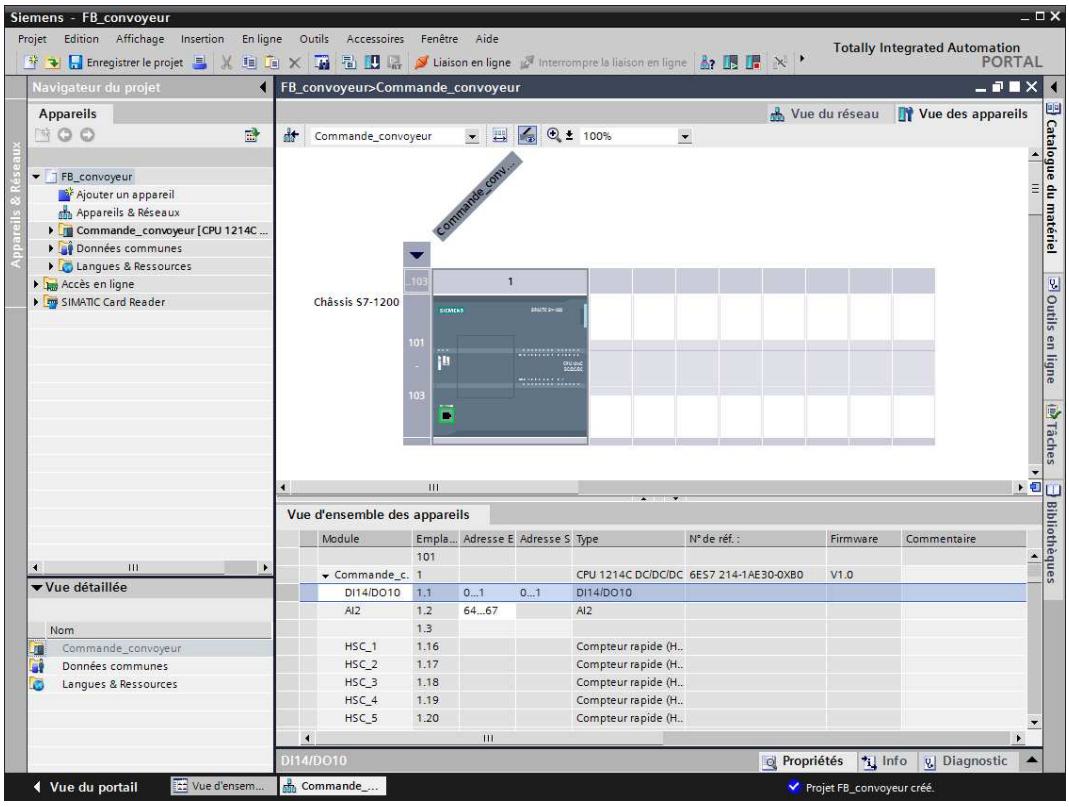
3. « **Mise en route** » est recommandée pour le début de la création du projet.
   Premièrement, nous voulons « **Configurer un appareil** » (« **Mise en route** > **Configurer un appareil** »).



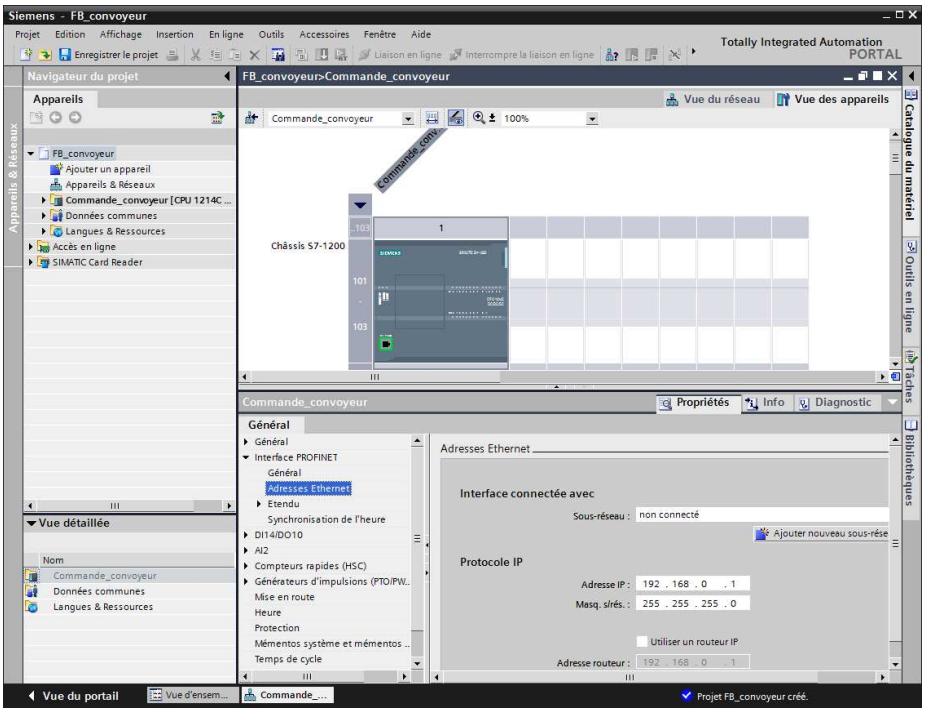
4. Puis « **Ajouter un appareil** » avec le nom d'appareil : *Commande_convoyeur.* Choisissez alors dans le catalogue la « **CPU 1214C** » avec la bonne combinaison de lettres derrière. (« **Ajouter un appareil** > **SIMATIC PLC** > **CPU 1214C** > **6ES7 214-1AG40-0XB0** > **Ajouter** »)



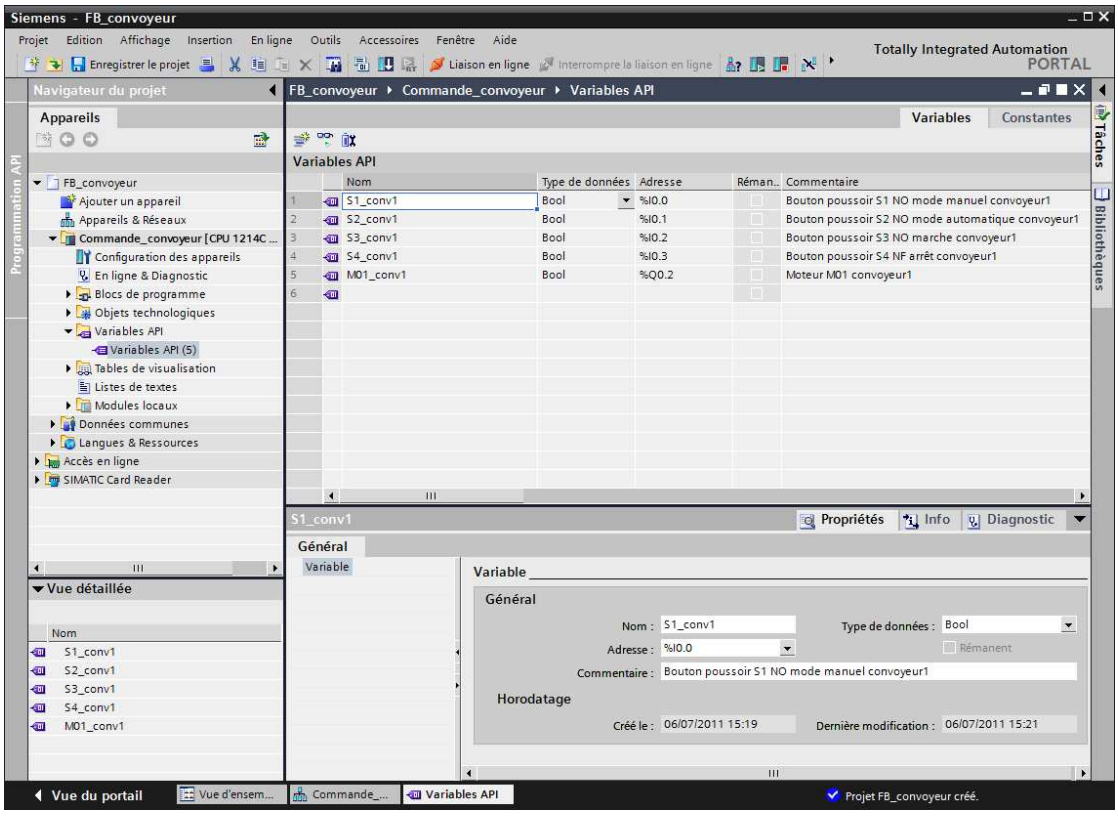
5. Le logiciel bouge automatiquement vers la vue du projet avec la configuration matérielle ouverte. Ici, on peut ajouter des modules supplémentaires depuis le **Catalogue du matériel** (fenêtre de droite), et dans la **Vue d'ensemble des appareils**, les adresses d'entrée/sortie peuvent être visualisées. Dans notre cas, les entrées intégrées à la CPU ont des adresses allant de %I 0.0 à %I 1.5 (soit 14 entrées) et les sorties intégrées des adresses allant de %Q 0.0 à %Q 1.1 (soit 10 sorties).



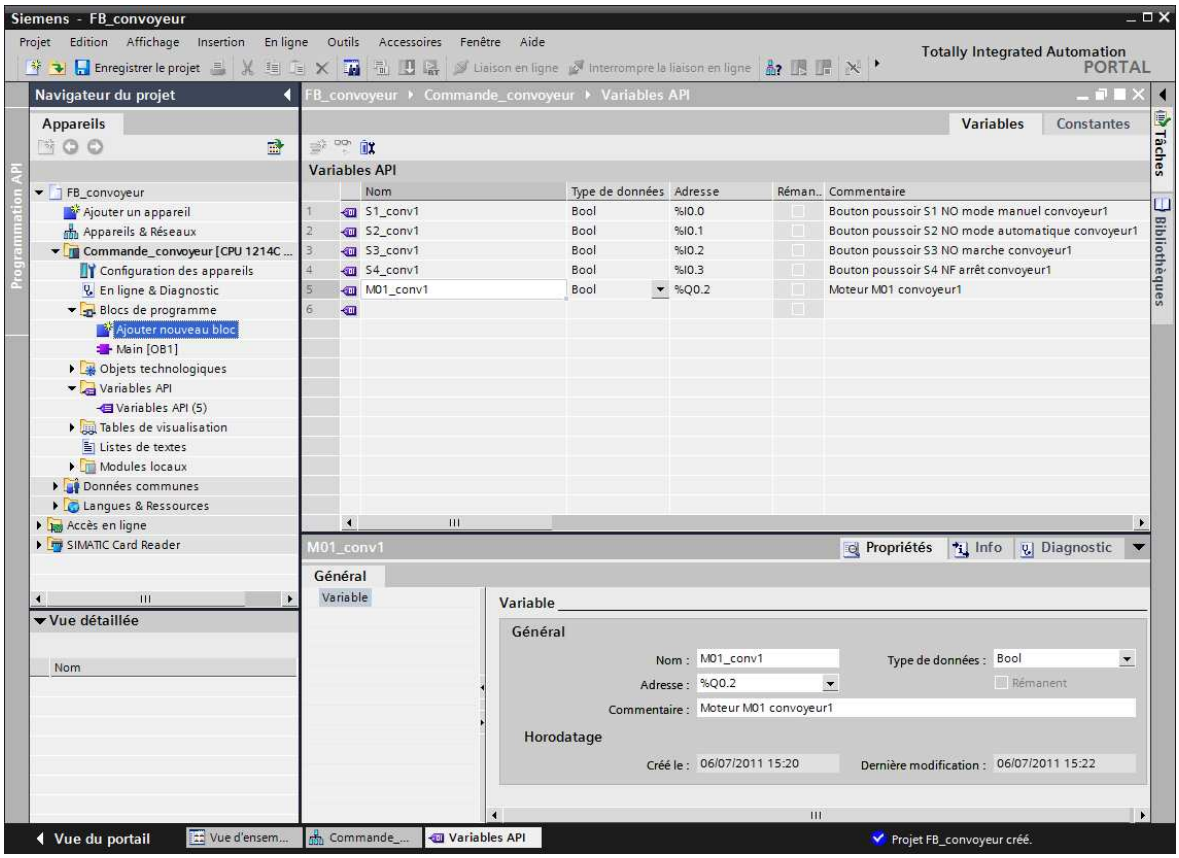
6. Afin que le logiciel puisse accéder dans la suite à la bonne CPU, son adresse IP et le masque de sous-réseau doivent être paramétrés (« **Propriétés** > **Général** > **Interface PROFINET** > **Adresses Ethernet** > **Adresse IP** : ***voir TP1*** et **Masq. s/rés.** : ***voir TP1*** »).



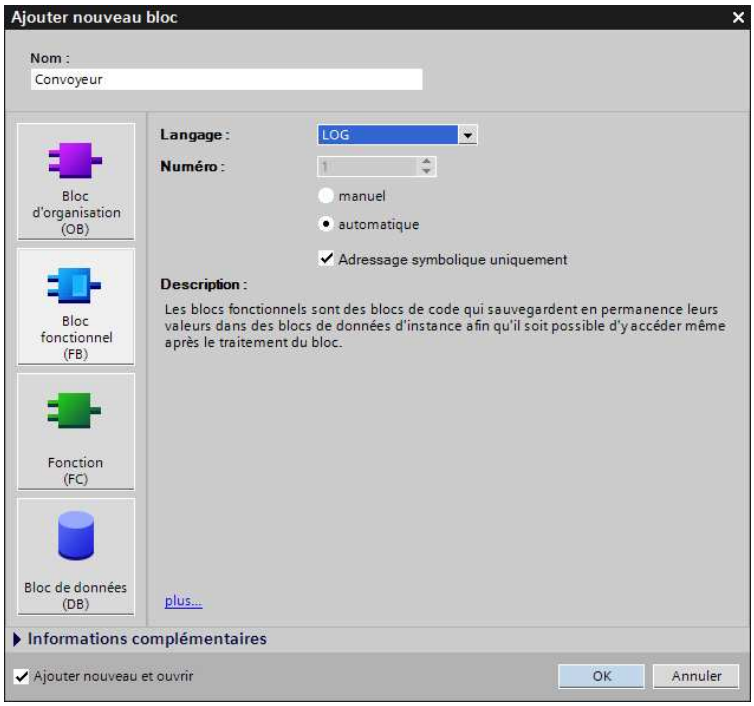
7. Puisque de nos jours on programme avec des variables plutôt qu'avec des adresses absolues, on doit spécifier les **variables globales de l'API**.
   Ces variables API globales sont des noms descriptifs et des commentaires pour ces entrées et sorties utilisées dans le programme. Plus tard, pendant la programmation, on pourra accéder à ces variables API via leurs noms.
   Ces variables globales peuvent être utilisées partout dans le programme, dans tous les blocs.
   À cette fin, sélectionnez dans le navigateur du projet « **Controle_citerne [CPU 1214C DC/DC/DC]** » puis « **Variables API** ». Avec un double-clic, ouvrez la table des variables API et entrez, comme montré ci-dessous, les noms des entrées et des sorties.



8. Pour créer le bloc fonctionnel FB1, sélectionnez dans le navigateur du projet « **Commande_convoyeur [CPU 1214 C DC/DC/DC]** » puis « **Blocs de programme** ». Double-cliquez ensuite sur « **Ajouter nouveau bloc** ».



9. Dans la nouvelle fenêtre, choisissez « **Bloc fonctionnel (FB)** » et donnez-lui le nom « *Convoyeur* ». Comme langage de programmation, choisissez « **LOG** » (logigramme). La numérotation est automatique. Puisque FB1 est appelée de toute façon par son nom symbolique, le numéro n'a plus beaucoup d'importance. Acceptez les saisies avec « **OK** ».



10. Le bloc « **Convoyeur [FC1]** » s'ouvre automatiquement. Avant de pouvoir écrire le programme, cependant, on doit déclarer les variables locales, qui ne sont connues que dans le bloc.

    Les variables sont divisées en 2 groupes :

- **Les paramètres qui forment l'interface du bloc pour les appels dans le programme :**

Type	Nom	Fonction	Reconnu dans
Paramètres d'entrée	Input	Paramètre dont la valeur est lue par le bloc	Fonctions, blocs de fonction et quelques types de blocs d'organisation
Paramètres de sortie	Output	Paramètre dont la valeur est écrite par le bloc	Fonctions et blocs de fonction
Paramètres d'entrée/sortie	InOut	Paramètre dont la valeur est lue par le bloc quand elle est appelée, et qui après traitement est écrite dans le même paramètre	Fonctions et blocs de fonction

- **Les données locales utilisées pour un stockage des résultats intermédiaires :**

Type	Nom	Fonction	Reconnu dans
Données locales temporaires	Temp	Variables utilisées pour un stockage temporaire des résultats intermédiaires. Les données temporaires sont conservées pour un cycle seulement	Fonctions, blocs de fonction et blocs d'organisation
Données locales statiques	Static	Variables utilisées pour un stockage statique des résultats intermédiaires dans le bloc de données d'instance. Les données statiques sont conservées jusqu'à leur réécriture, soit pour plusieurs cycles	Fonctions et blocs de fonction

11. <u>Déclarons maintenant les variables locales nécessaires pour notre exemple :</u>

Input :

manuel	Le bouton de sélection du mode Manuel est entré ici (contact NO)
auto	Le bouton de sélection du mode Automatique est entré ici (contact NO)
marche	Le bouton de mise en marche est entré ici (contact NO)
arret	Le bouton d'arrêt est entré ici (contact NF)

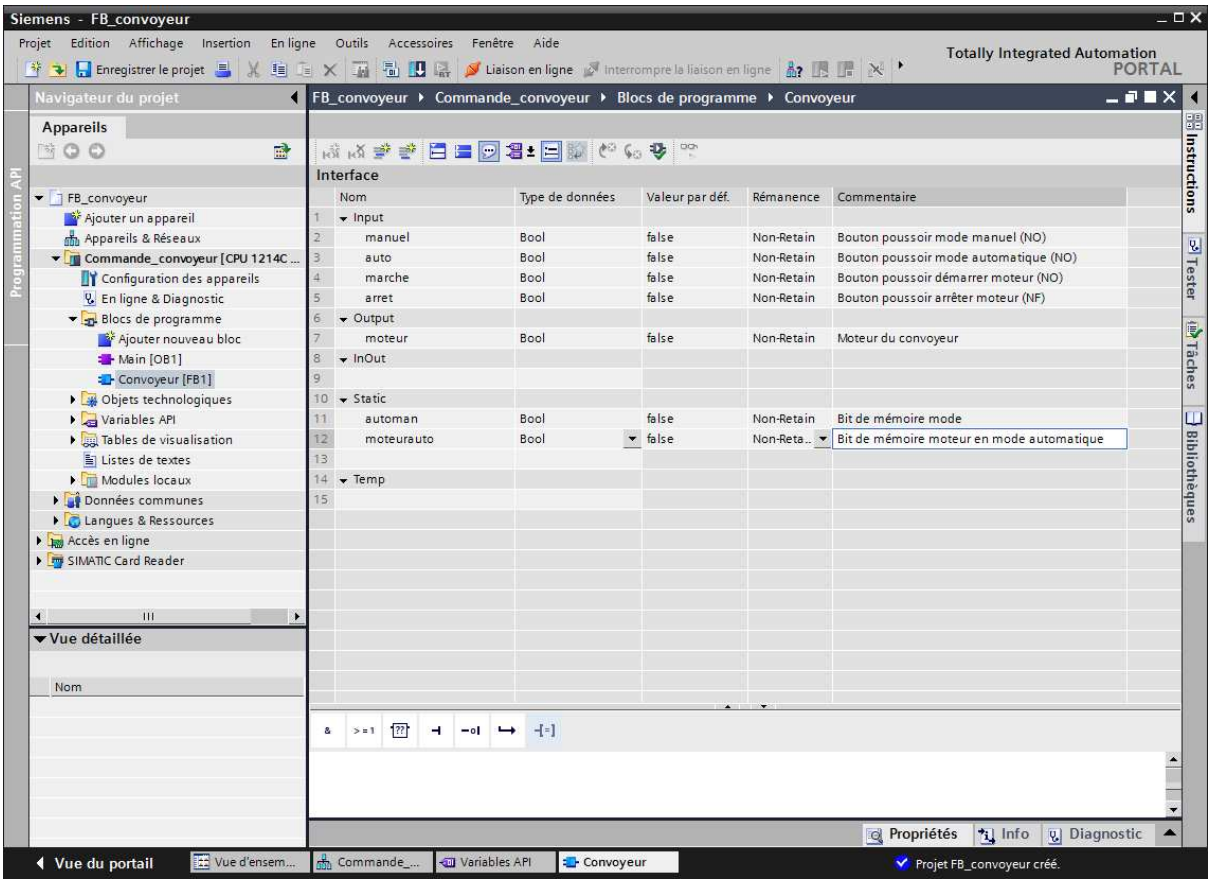
Output :

moteur	L'état de la sortie de commande du moteur est écrite ici

Static :

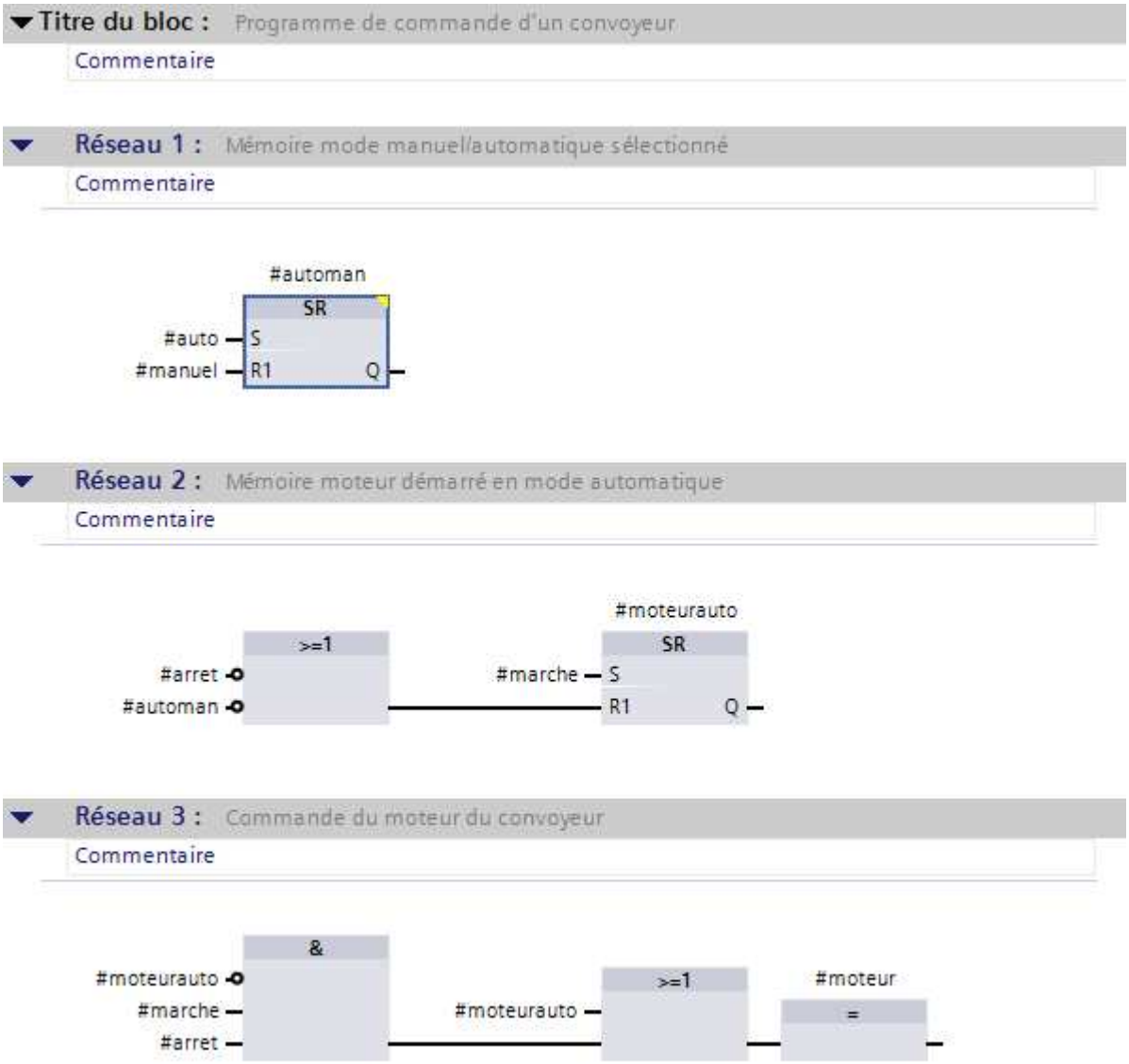
automan	Le mode de fonctionnement présélectionné est stocké ici
moteurauto	On stocke ici si le moteur du convoyeur a été mis en mode auto

Toutes les variables de cet exemple sont de type « Bool », ce qui veut dire qu'elles ne peuvent prendre que les valeurs « 1 » (vrai) ou « 0 » (faux).
Dans cet exemple, il est important de noter que l'état des deux variables « automan » et « moteurauto » doit être conservé pendant une plus grande période de temps. Pour cette raison, le type de variable « **Static** » doit être utilisé ici. Ce type de variable n'existe que dans les blocs fonctionnels FB.
Pour une meilleure compréhension, il est préférable d'écrire des commentaires pour chaque variable.

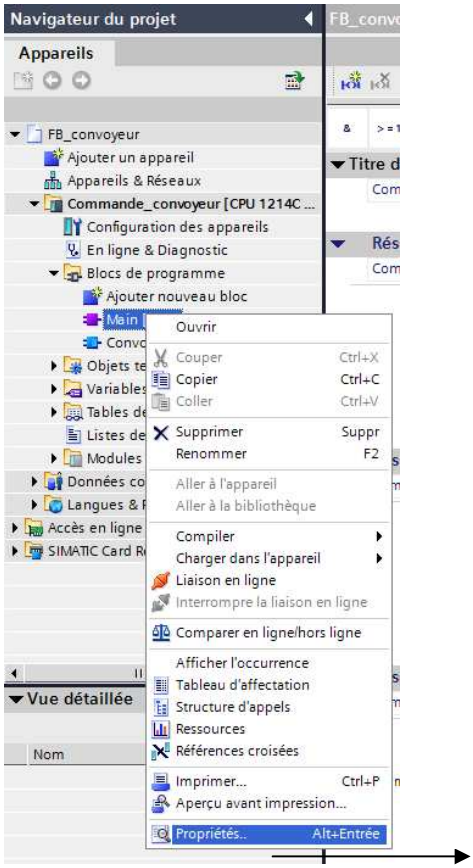


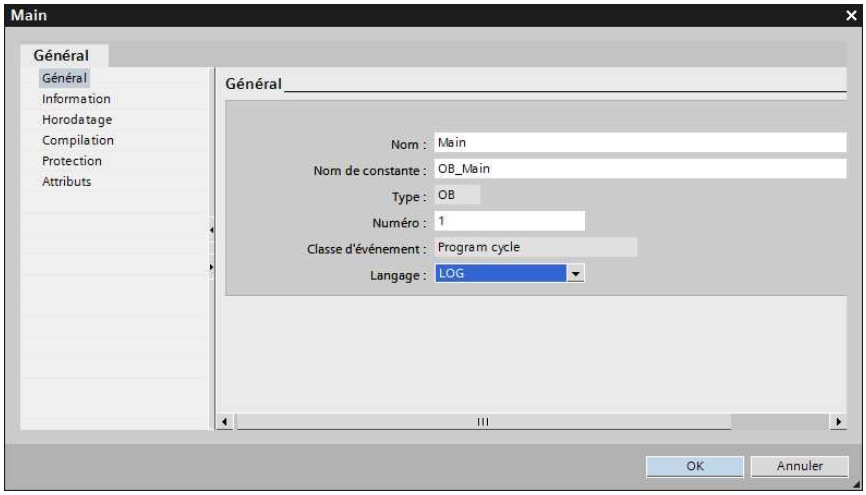
12. Après avoir déclaré les variables locales, on peut écrire le programme en utilisant les noms de variables (les variables sont identifiées par le symbole « # »). Avec les blocs logiques, par exemple, ça donnerait :

    Attention de ne pas oublier les négations de certaines entrées !!

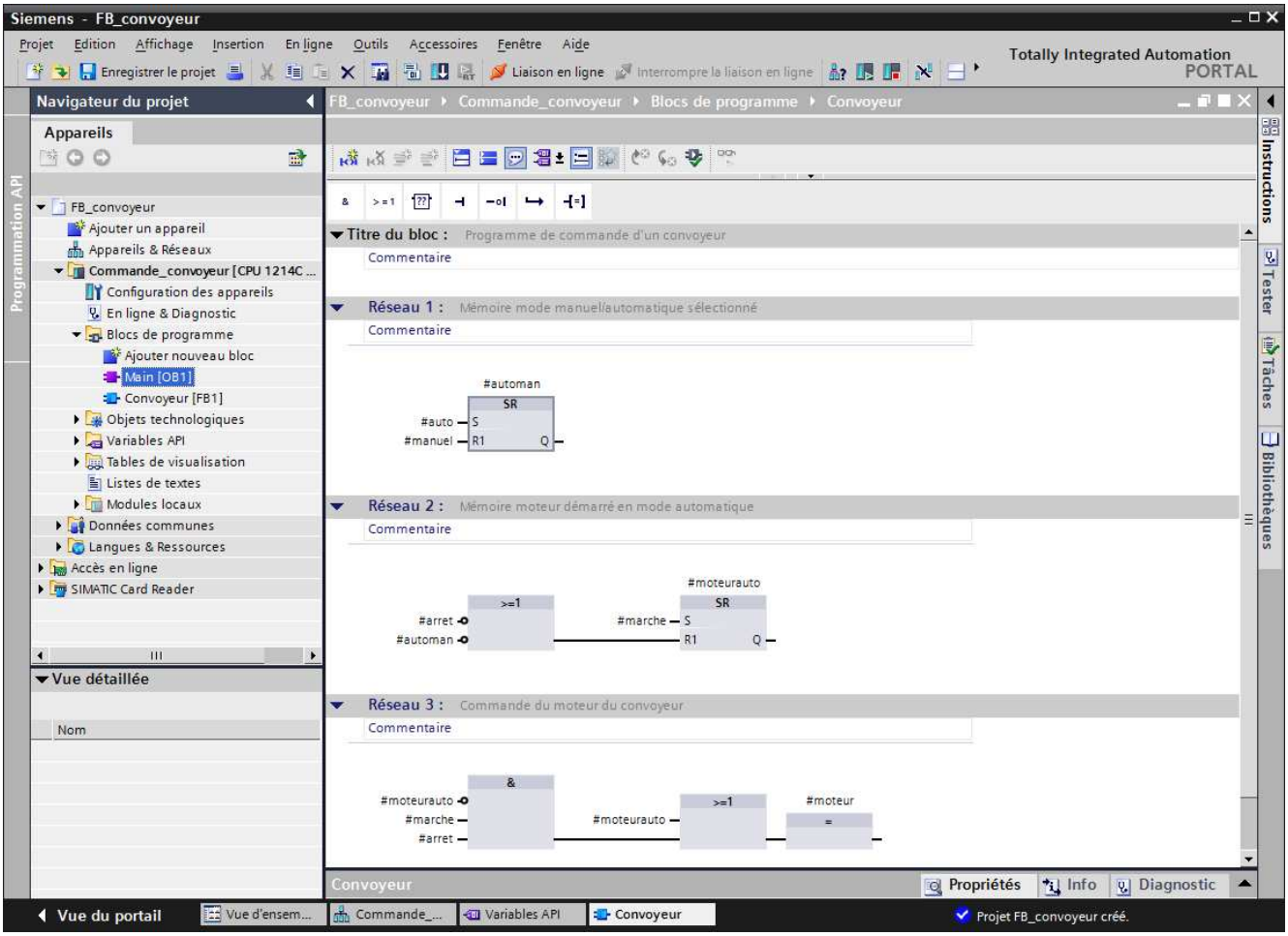


13. Ensuite, faites un clic droit sur le bloc « **Main [OB1]** » (bloc traité de façon cyclique) et cliquez sur « **Propriétés..** ».
    Là, changez le langage de programmation en « **LOG** », puis confirmez avec « **OK** ».

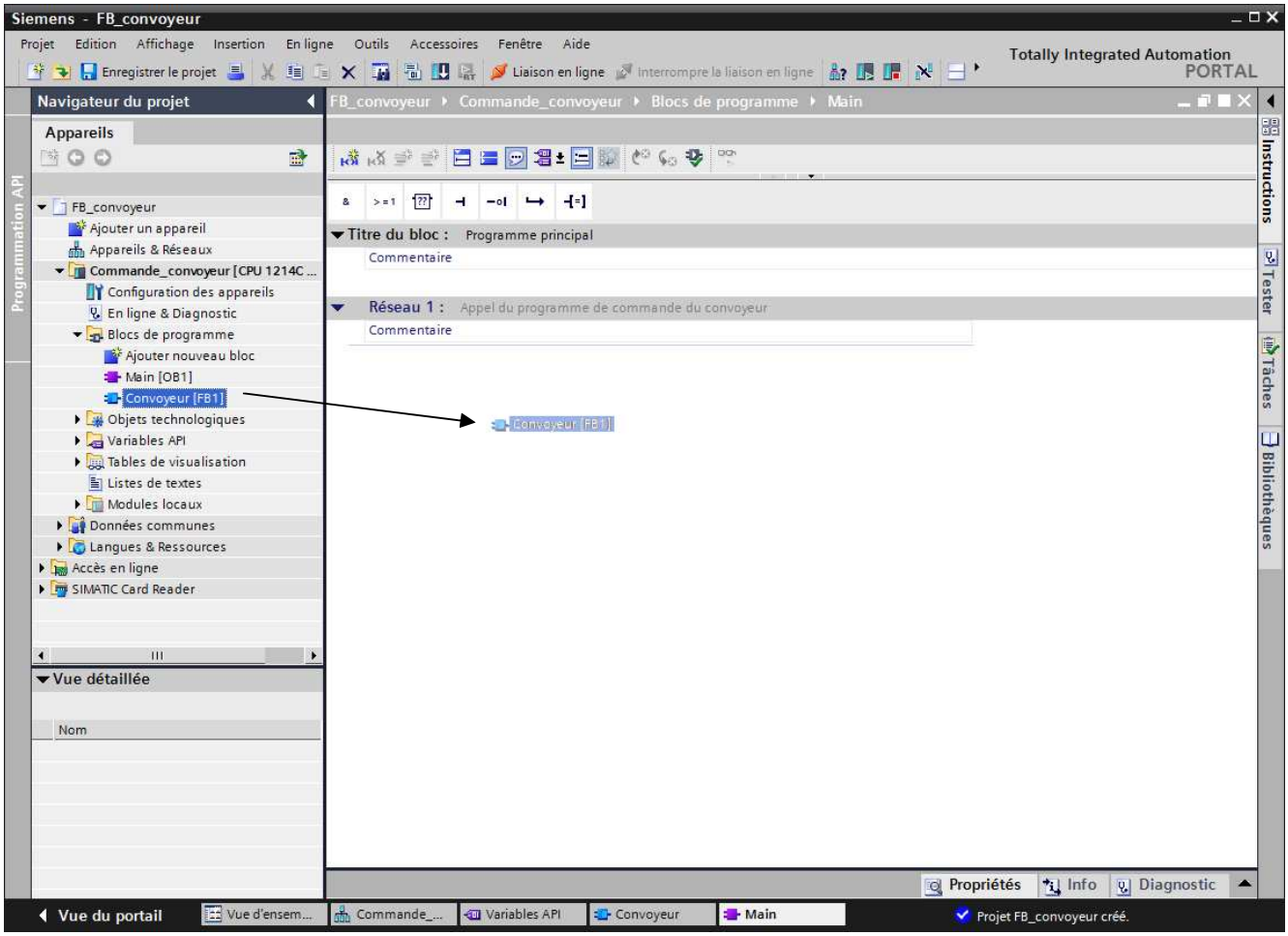




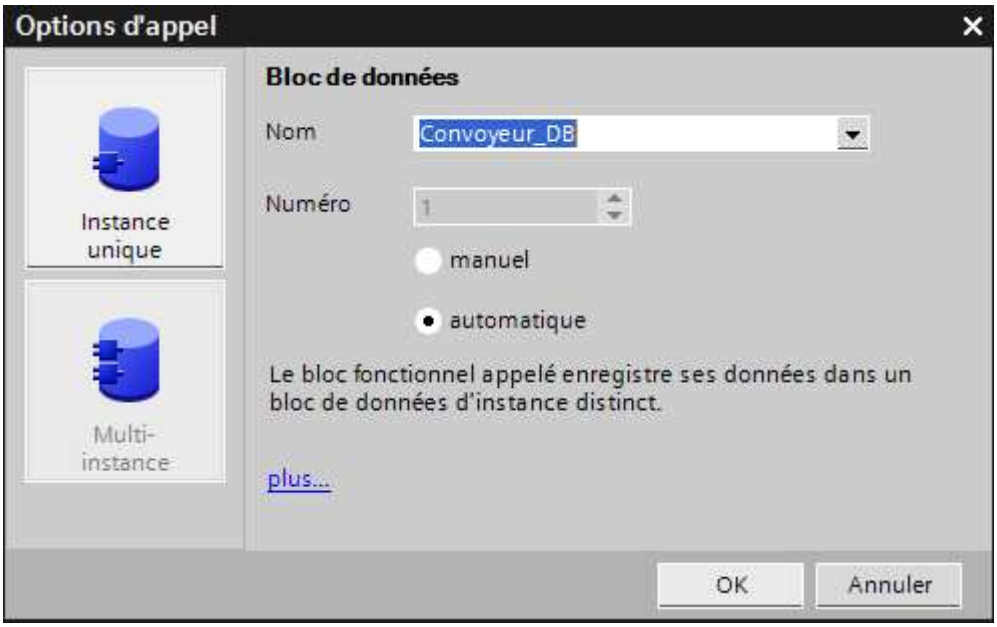
14. Maintenant, on va appeler le bloc « *Convoyeur [FC1]* » dans le bloc « *Main [OB1]* », sinon le bloc ne serait pas traité du tout. Double-cliquez sur « **Main [OB1]** » pour l'ouvrir.



15. À l'aide d'un glisser-déposer, déplacez le bloc « **Convoyeur [FC1]** » dans le réseau 1 du bloc « **Main [OB1]** ». Rappelez-vous aussi de bien documenter les réseaux du bloc *Main*, de la même manière que dans *Convoyeur*.



16. Puisque nous sommes en train de travailler avec un bloc fonctionnel, une mémoire doit lui être attribuée. Dans le SIMATIC S7-1200, les blocs de données font office de cette mémoire. Ce bloc de données ainsi affecté est appelé « **Bloc de données d'instance** ».
    Ici, il est défini et généré automatiquement, cliquez simplement sur « **OK** ».

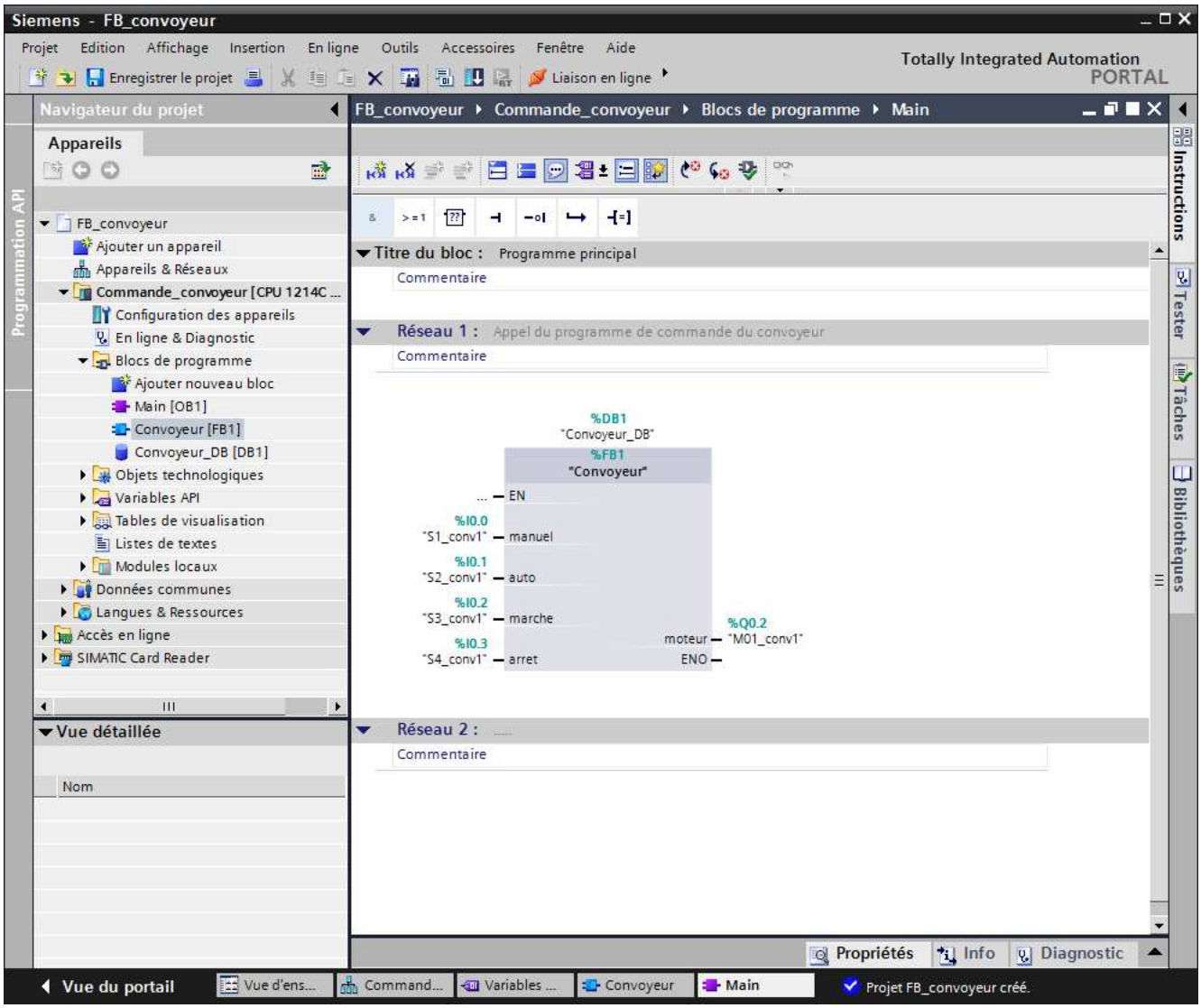


Indication :

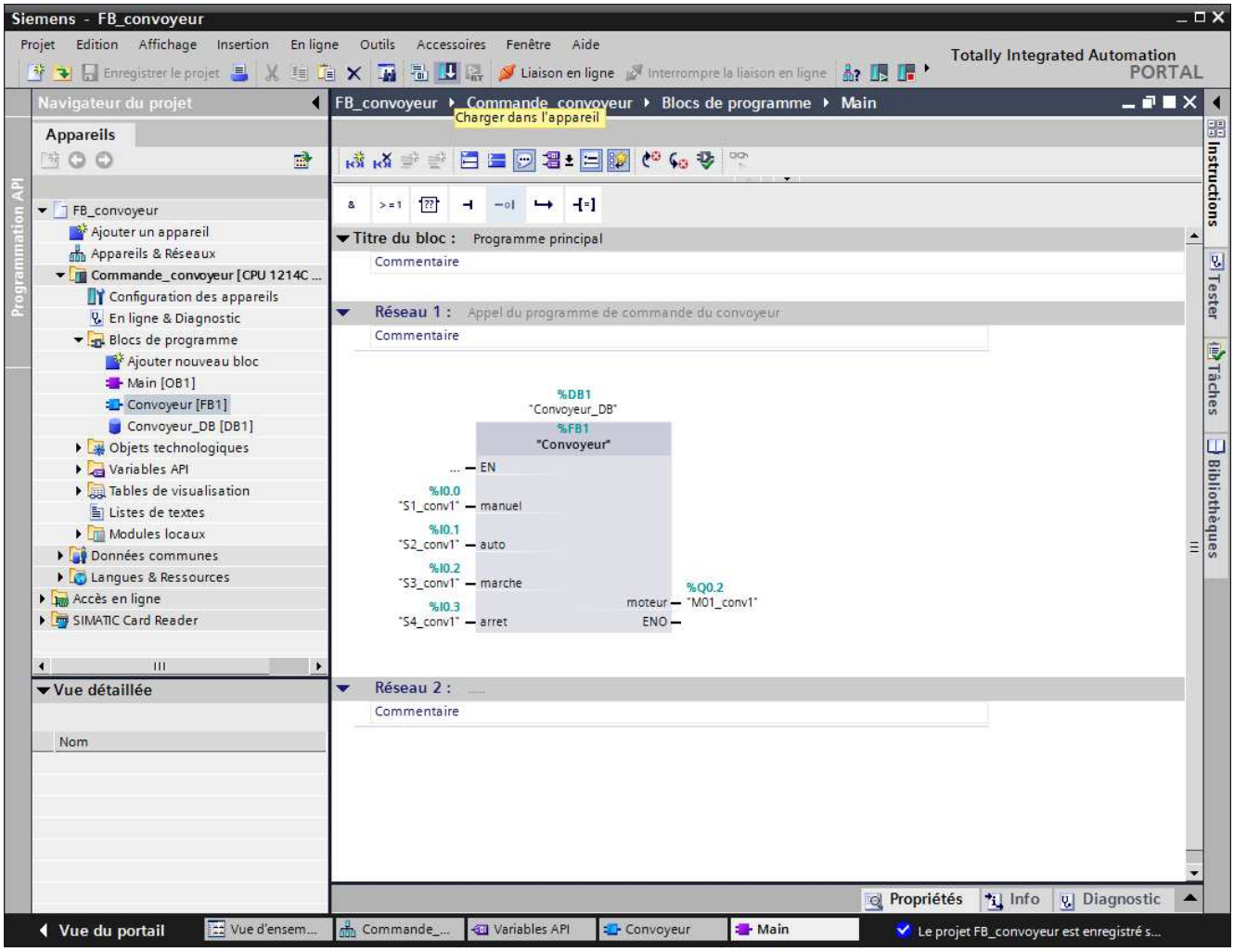
Des exemples supplémentaires sur la gestion et la création d'instances sont données dans le TP3.

17. Maintenant, on connecte les variables d'entrée et de sortie dans OB1 avec les variables API comme indiqué ci-dessous.

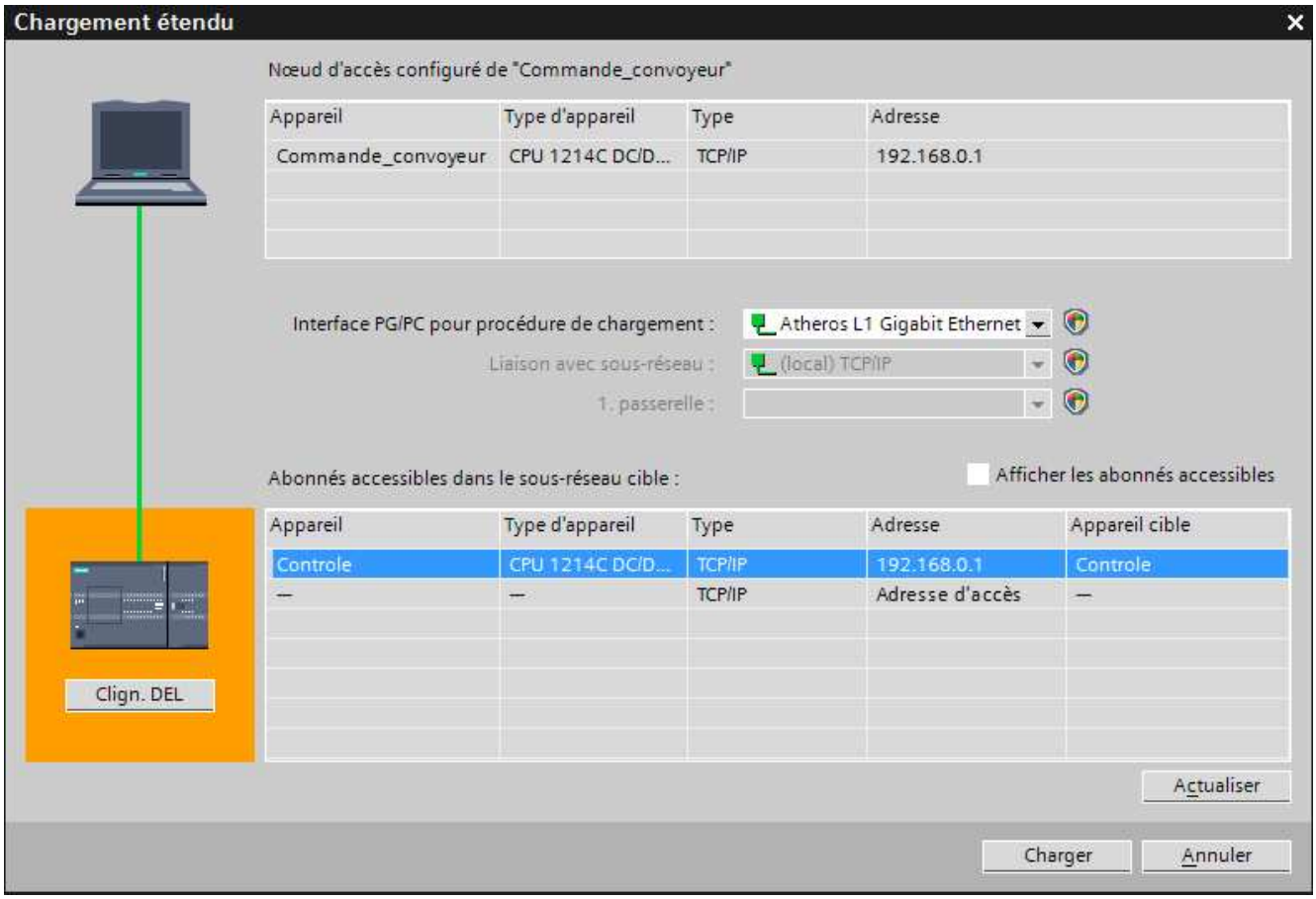
    Il est également temps de sauvegarder votre projet, en cliquant sur : Enregistrer le projet.



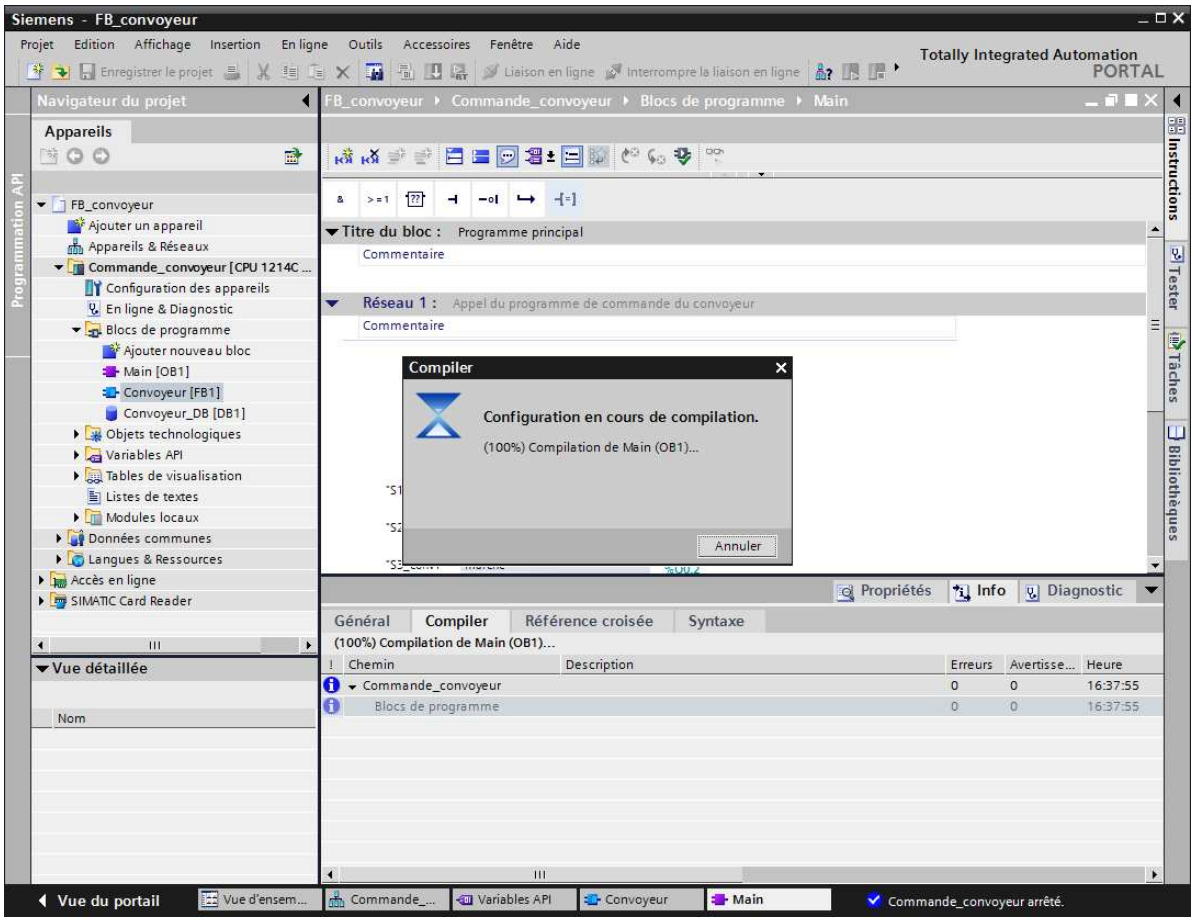
18. Pour charger votre programme entier dans la CPU, surlignez d'abord « **Commande_convoyeur [CPU 1214C DC/DC/DC]** » en cliquant une fois dessus. Cliquez ensuite sur le symbole « **Charger dans l'appareil** ».



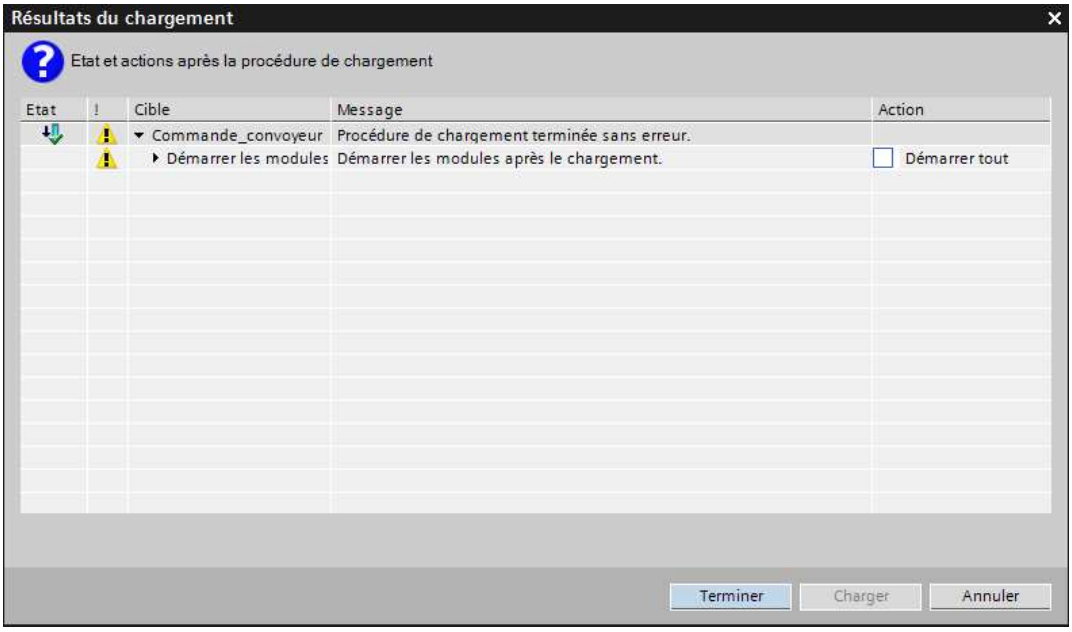
19. Dans le cas où vous auriez oublié de paramétrer l'interface PG/PC auparavant, une fenêtre où il est encore possible de le faire s'ouvre.



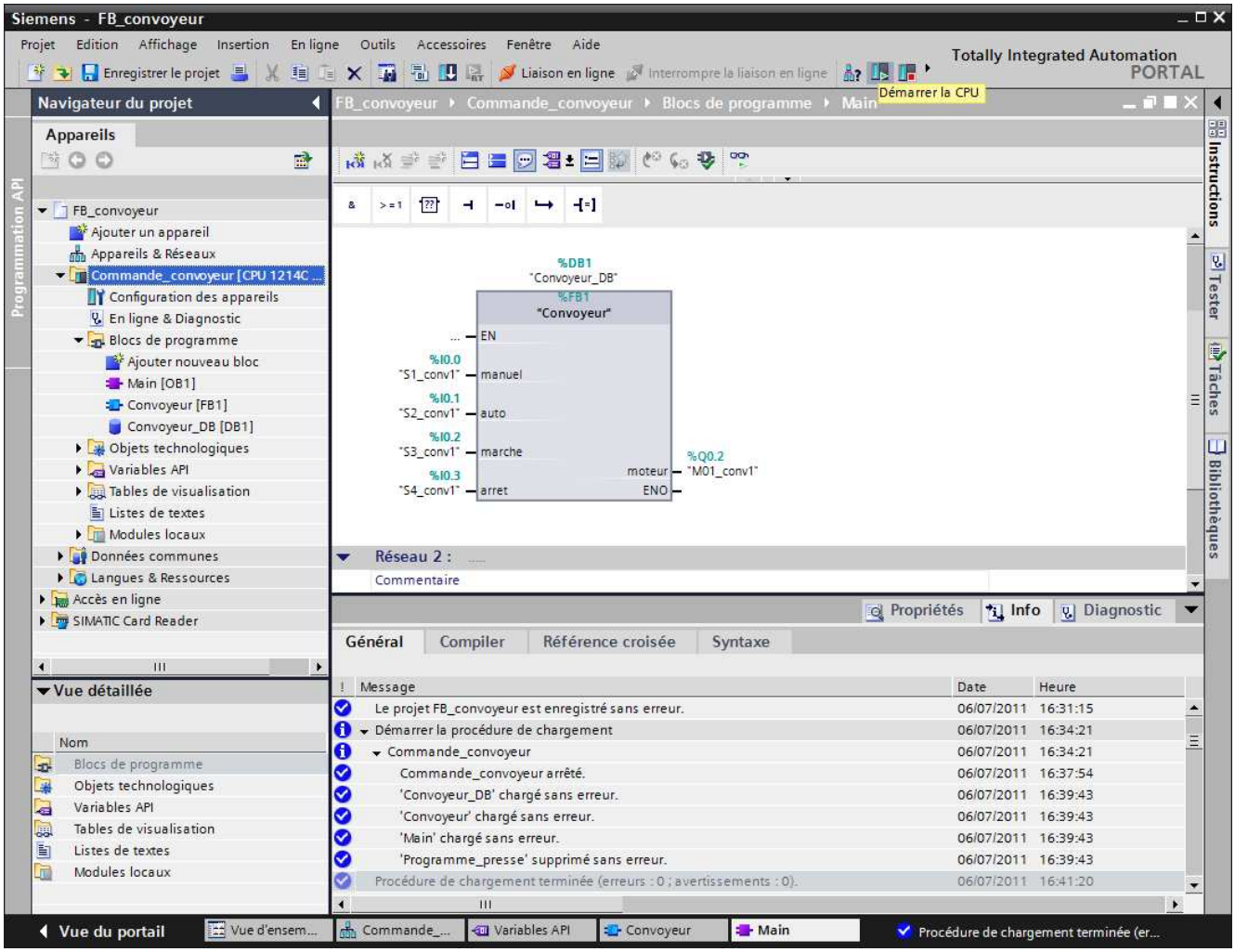
20. Pendant la compilation, l'état de progression est affiché dans une fenêtre.



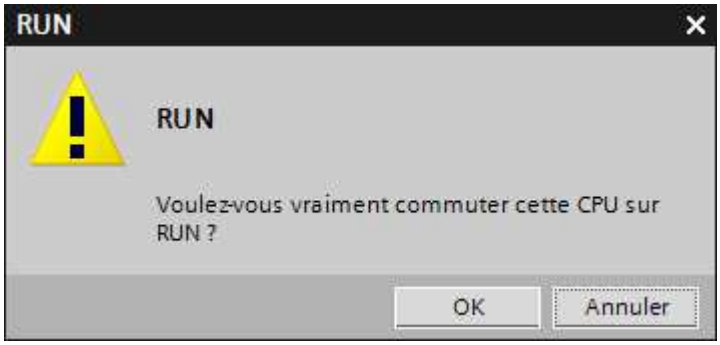
21. Si la compilation s'est correctement déroulée, cela s'affiche dans la fenêtre. Cliquez maintenant sur « **Charger** », puis « **Terminer** ».



22. Ensuite, démarrez la CPU en cliquant sur le symbole  « **Démarrer la CPU** ».

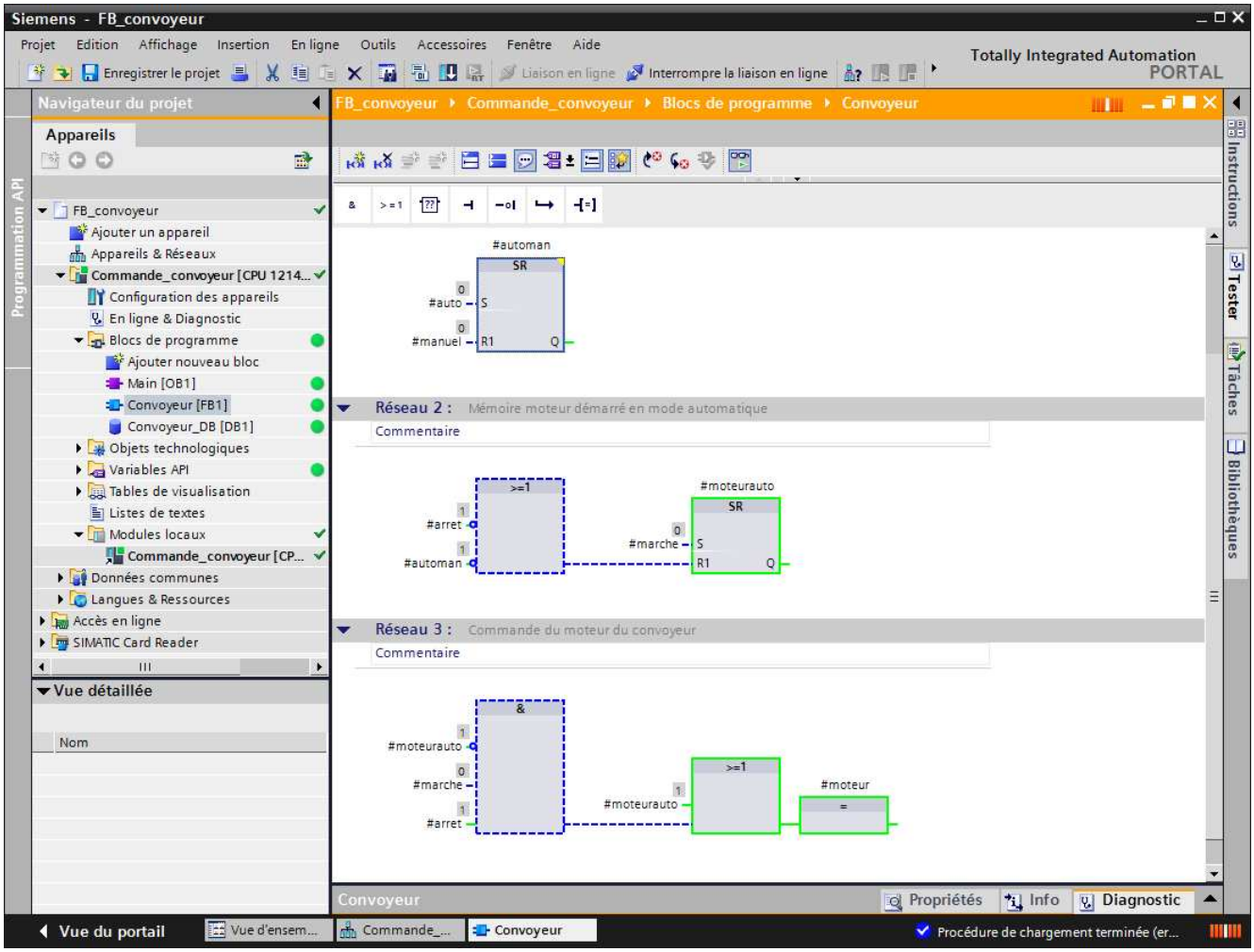


23. Confirmez le fait que vous vouliez vraiment commuter la CPU sur *RUN* en cliquant sur « **OK** ».



24. En cliquant sur l'icône  « **Activer/désactiver visualisation du programme** », il est possible de surveiller l'état des variables sur le bloc « *Convoyeur* » pendant que vous testez le programme en commutant les interrupteurs de la maquette.

    Remarquez que la fenêtre « *Navigateur du projet* » est devenue orange, ce qui signifie que vous travaillez désormais en ligne avec l'automate.



25. Puisque notre bloc « *Convoyeur* » a été créé selon les règles des blocs standards (pas d'utilisation de variables globales dans le bloc !), il peut être utilisé et appelé un nombre quelconque de fois.
    Ci-dessous, on a rajouté des variables dans la table des variables API, avec les entrées et les sorties pour deux convoyeurs.

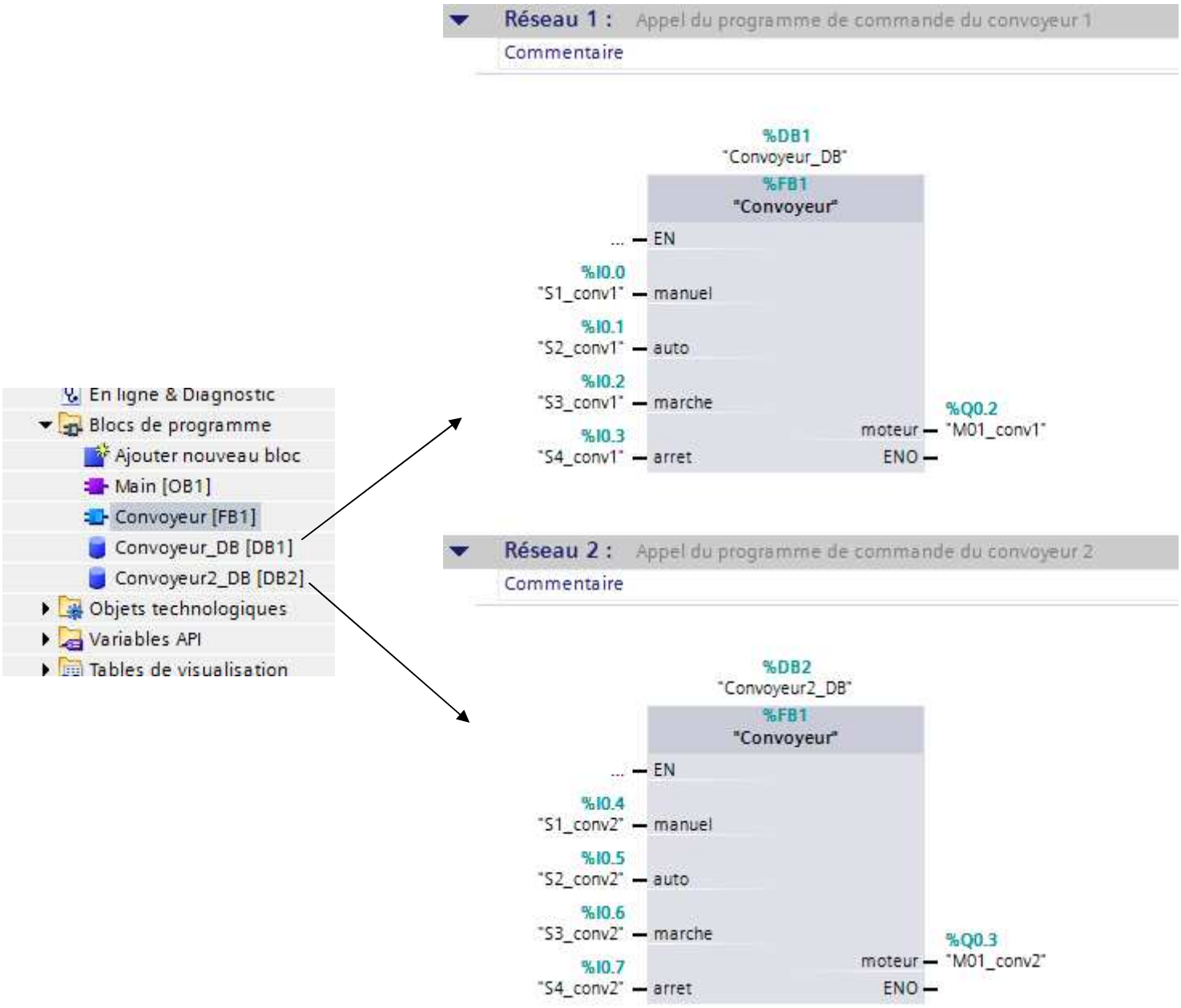
FB_convoyeur ▸ Commande_convoyeur ▸ Variables API

Variables | Constantes

Variables API

	Nom	Type de données	Adresse	Réman..	Commentaire
1	S1_conv1	Bool	%I0.0		Bouton poussoir S1 NO mode manuel convoyeur1
2	S2_conv1	Bool	%I0.1		Bouton poussoir S2 NO mode automatique convoyeur1
3	S3_conv1	Bool	%I0.2		Bouton poussoir S3 NO marche convoyeur1
4	S4_conv1	Bool	%I0.3		Bouton poussoir S4 NF arrêt convoyeur1
5	M01_conv1	Bool	%Q0.2		Moteur M01 convoyeur1
6	S1_conv2	Bool	%I0.4		Bouton poussoir S1 NO mode manuel convoyeur2
7	S2_conv2	Bool	%I0.5		Bouton poussoir S2 NO mode automatique convoyeur2
8	S3_conv2	Bool	%I0.6		Bouton poussoir S3 NO marche convoyeur2
9	S4_conv2	Bool	%I0.7		Bouton poussoir S4 NO arrêt convoyeur2
10	M01_conv2	Bool	%Q0.3		Moteur M01 convoyeur2
11					

26. Maintenant, le bloc « *Convoyeur* » peut être appelé deux fois dans OB1. Les 2 blocs voient alors leurs bornes connectées respectivement avec des variables différentes. Pour chaque appel, un autre bloc de données d'instance est défini.


En ligne & Diagnostic
Blocs de programme
Ajouter nouveau bloc
Main [OB1]
Convoyeur [FB1]
Convoyeur_DB [DB1]
Convoyeur2_DB [DB2]
Objets technologiques
Variables API
Tables de visualisation
Réseau 1 : Appel du programme de commande du convoyeur 1
Commentaire
%DB1
"Convoyeur_DB"
%FB1
"Convoyeur"
... EN
%I0.0
"S1_conv1" manuel
%I0.1
"S2_conv1" auto
%I0.2
"S3_conv1" marche
%I0.3
"S4_conv1" arret
moteur %Q0.2 "M01_conv1"
ENO
Réseau 2 : Appel du programme de commande du convoyeur 2
Commentaire
%DB2
"Convoyeur2_DB"
%FB1
"Convoyeur"
... EN
%I0.4
"S1_conv2" manuel
%I0.5
"S2_conv2" auto
%I0.6
"S3_conv2" marche
%I0.7
"S4_conv2" arret
moteur %Q0.3 "M01_conv2"
ENO