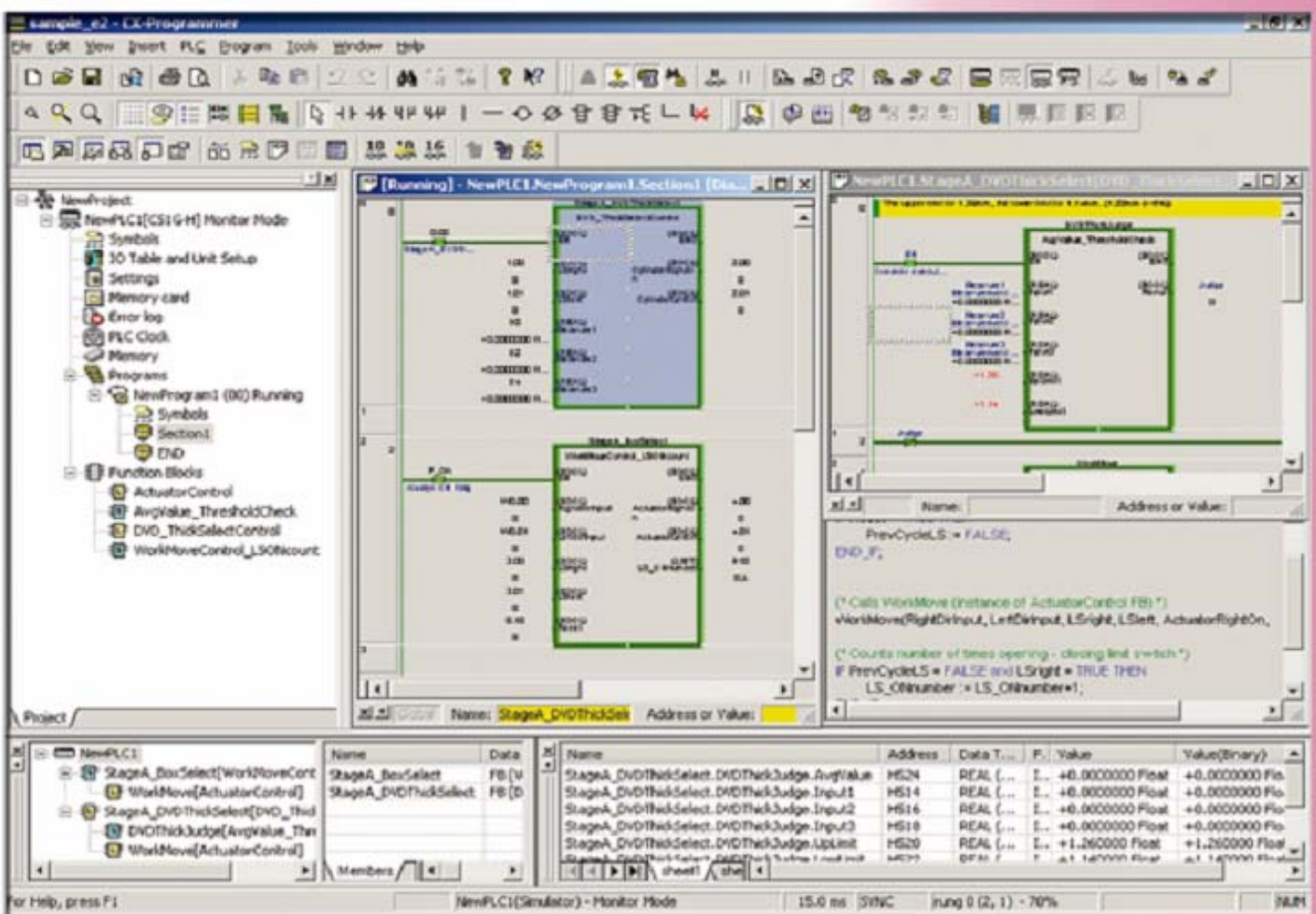
Blocs Fonction

Guide d'introduction



Cat. No. R133-FR2-02

Le CD-ROM de CX-One / CX-Programmer contient un manuel d’utilisation au format PDF. Avant d’utiliser ce produit, veuillez lire les sections Introduction, Consignes de sécurité et Précautions d’utilisation. Le Guide d’implémentation des blocs de fonctions (Function Block Implementation Guide) décrit les opérations de base permettant d’utiliser la bibliothèque FB OMRON et fournit des conseils pour créer un programme utilisateur à l’aide de blocs de fonctions. Les consignes et les explications détaillées figurent dans l'aide et dans le manuel au format PDF.

* Acrobat Reader version 4.0 ou ultérieure est nécessaire pour lire le fichier PDF.

Table des matières

Introduction

Ce document fournit des conseils relatifs à l'utilisation de la bibliothèque FB OMRON et à la création de blocs de fonctions (FB), disponibles pour les UC SYSMAC série CS1/CJ1-H/CJ1M d'Omron (version 3.0 ou ultérieure) et CX-Programmer version 5.0 ou ultérieure.

Nouvelles fonctions disponibles dans CX-Programmer version 6.0

Imbrication de blocs de fonctions

L'imbrication de blocs de fonctions (FB, Function Block) simplifie désormais l'organisation et la réutilisation des programmes utilisateur. En effet, les blocs de fonctions peuvent être appelés à partir d'un programme de texte structuré (ST, Structured Text) de programmes converti en FB. Les fonctions ci-dessous sont également prises en charge à cette fin.

Compréhension simplifiée de la structure des programmes... Visualiseur d'exemple FB
Gestion de composants, notamment les FB appelés... Enregistrement et chargement de fichiers, notamment les FB appelés
Accès rapide aux FB appelés... Double-clic sur une instance FB (appel d'instruction)

Surveillance de schéma FB

A l'instar du programme principal, il est possible de surveiller l'état du programme FB.

Tableau de références croisées dans le schéma FB

A l'instar du programme principal, le tableau de références croisées est à présent disponible dans le programme FB.
En outre, il est à présent possible d'accéder à la bobine de sortie depuis le contact à l'aide de la barre d'espace.

Accès à l'aide ST

Dans l'Editeur texte structuré, vous pouvez accéder à une rubrique d'aide à partir d'un menu contextuel afin de vérifier la syntaxe en toute simplicité pour la programmation ST.

Accès aux références de la bibliothèque FB OMRON

Vous pouvez afficher un fichier PDF des références croisées de bibliothèque qui décrivent les caractéristiques d'une bibliothèque FB OMRON enregistrée dans un fichier de projet.

Attention :

Vous pouvez utiliser un programme contenant des FB imbriqués pour une UC série CS1/CJ1-H/CJ1M (version 3.0 ouultérieure). Toutefois, si vous tentez de télécharger un programme contenant des FB imbriqués à l'aide de CX-Programmer (version 5.0 ou antérieure), qui ne prend pas en charge l'imbrication, il en résulte un échec ou un état incomplet. Si vous enregistrez le fichier tel quel, vous ne pourrez pas établir de distinction entre les programmes incomplets et les programmes corrects.

[CX-Programmer version 5.0]
Les messages suivants s'affichent après le téléchargement :
Des propriétés API non supportées par cette version de CX-Programmer sont définies dans l'API cible de connexion. Les propriétés API ne s'afficheront pas correctement. Voulez-vous continuer ?

[CX-Programmer version 4.0]
Le message suivant s'affiche après le téléchargement :
Bloc fonction ou données autres que le schéma contacts incluses dans les programmes.

[CX-Programmer version 3.x]
Après le téléchargement, le message Erreur de décompilation s'affiche et aucun programme ne s'affiche.

Nouvelles fonctions disponibles dans CX-Programmer version 6.1

Surveillance ST, exécution des étapes

Pour simplifier le débogage ST du langage de traitement séquentiel, les fonctions suivantes sont prises en charge :
Affichage et modification de la valeur actuelle pendant l'exécution du programme ST.
Arrêt de l'exécution au point d'interruption et exécution des étapes à l'aide de CX-Simulator

Fonction de protection des FB

Il est possible de masquer des FB pour éviter toute modification par inadvertance, fuite de savoir-faire et modification incorrecte du programme.

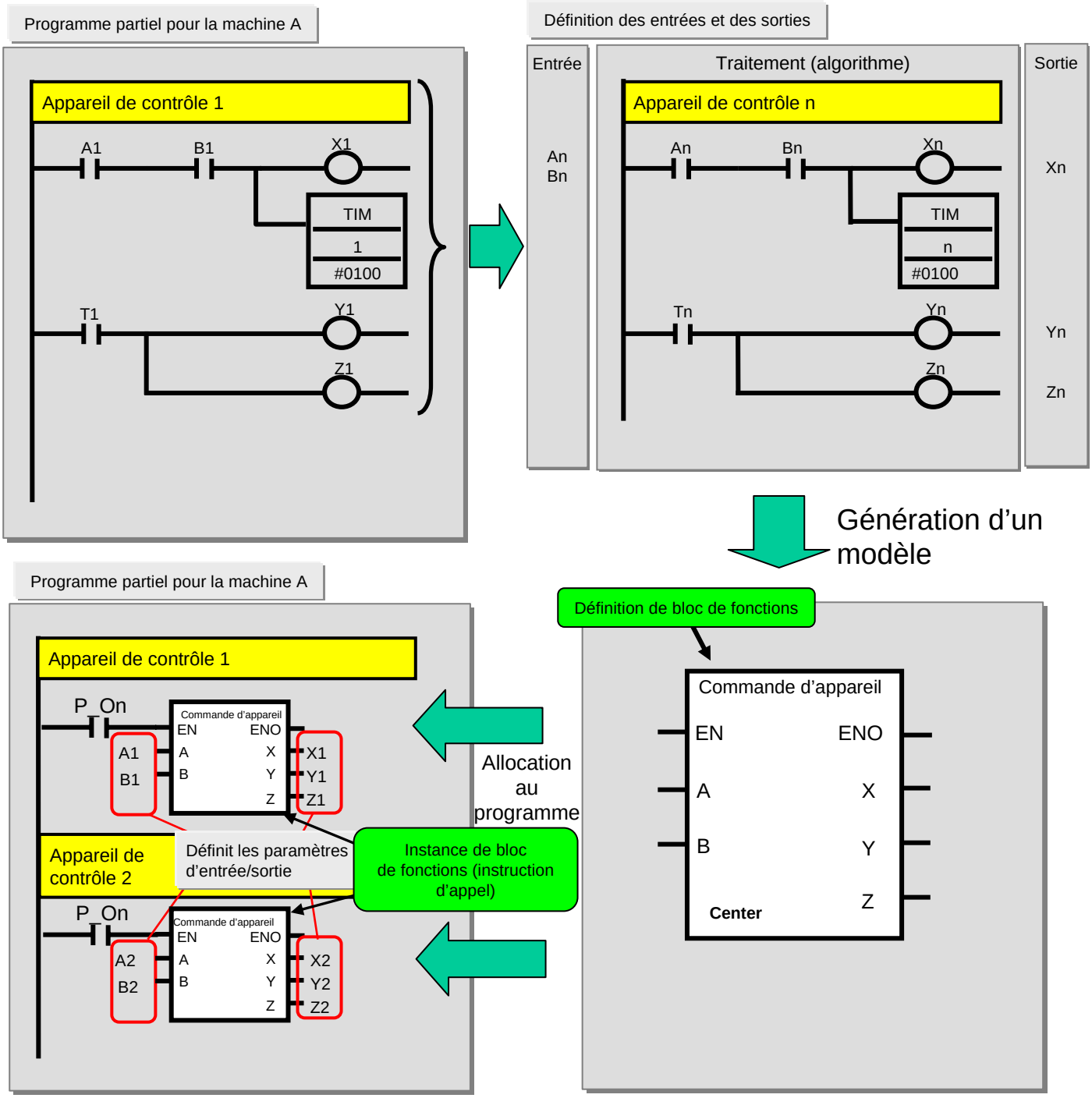
Chapître 1

Bibliothèque FB Omron

Function Block

1. Définition d'un bloc de fonctions

Un bloc de fonctions est un ensemble de programmes (ou fonctions) contenu dans un élément de programme pouvant être utilisé dans le schéma contact. Un élément de contact est nécessaire pour lancer la fonction. Toutefois, les entrées et les sorties peuvent être modifiées à l'aide de paramètres utilisés dans la disposition du schéma.
Il est possible de réutiliser les fonctions en tant que même élément (même mémoire) ou que nouvel élément disposant d'une mémoire propre.

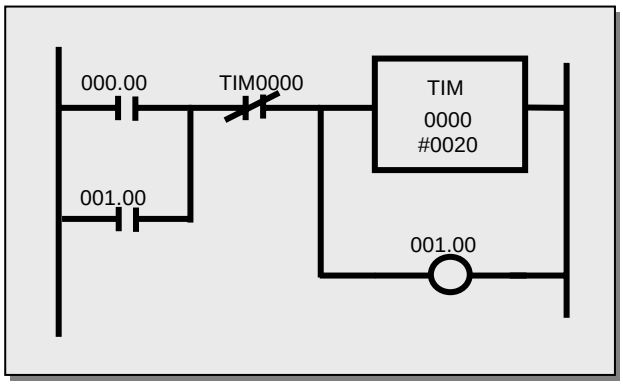


Définition de bloc de fonctions : contient la logique définie (algorithme) et l'interface d'E/S. Les adresses mémoire ne sont pas allouées dans la définition de bloc de fonctions.
Instance de bloc de fonctions (instruction d'appel) : instruction qui appelle l'instance de bloc de fonctions en cas d'utilisation par le programme, à l'aide de la mémoire allouée à l'instance.

2. Exemple d'un bloc de fonctions

Les schémas ci-dessous décrivent un exemple de bloc de fonctions correspondant à un circuit temporisé à utiliser dans le schéma. Il est possible de modifier le point de consigne de l'instruction TIM afin de réallouer l'heure définie pour désactiver la sortie du segment de contact. Grâce au bloc de fonctions ci-dessous, il est possible de rendre la limite temporelle du circuit arbitraire en ne modifiant qu'un seul paramètre.

Schéma de contact



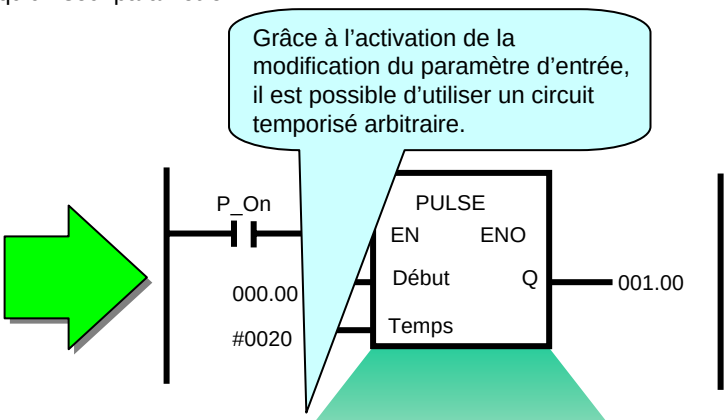
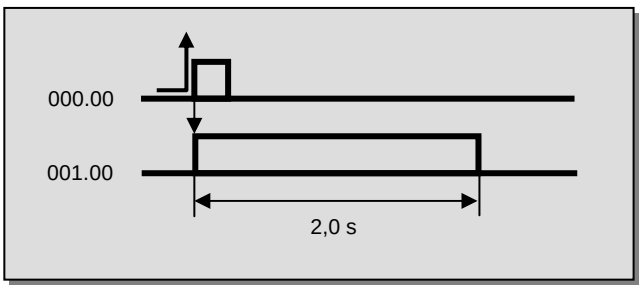
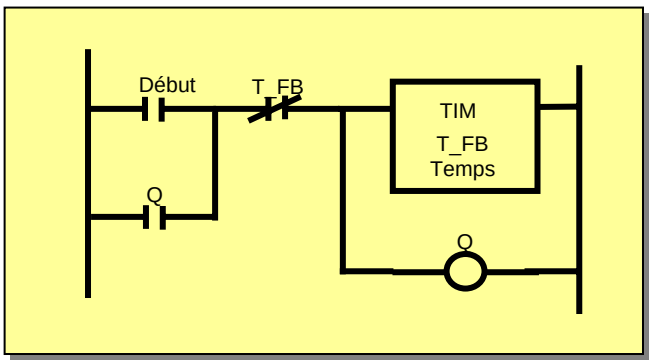


Schéma de temporisation





3. Présentation de la bibliothèque FB OMRON

La bibliothèque FB OMRON est un ensemble de fichiers de bloc de fonctions fournis par Omron. Ces fichiers permettent de simplifier les programmes et contiennent des fonctionnalités standard destinées à la programmation d'API et de fonctions de composants FA Omron.

3-1. Avantages de la bibliothèque FB OMRON

La bibliothèque FB OMRON est un ensemble d'exemples de blocs de fonctions qui visent à améliorer la connectivité des unités pour les API et les composants FA fabriqués par Omron. Voici les avantages dont vous bénéficiez lorsque vous utilisez la bibliothèque FB OMRON :

(1) Il n'est pas nécessaire de créer des schémas de contact à l'aide des fonctions de base des API et des composants FA

Vous pouvez consacrer plus de temps aux programmes personnalisés pour les appareils externes, car les schémas de contact de base sont déjà disponibles.

(2) Facile à utiliser

Pour obtenir un programme en ordre de marche, il suffit de charger le fichier de bloc de fonctions pour exécuter la fonctionnalité cible, puis d'entrer une instance (instruction d'appel d'un bloc de fonctions) dans le programme de schéma de contact et de définir les adresses (paramètres) des entrées et des sorties.

(3) Il n'est pas nécessaire de tester le fonctionnement du programme

Omron a testé la bibliothèque des blocs de fonctions. Par conséquent, il n'est plus nécessaire de déboguer les programmes permettant d'exploiter l'unité et les composants FA pour les API.

(4) Facile à comprendre

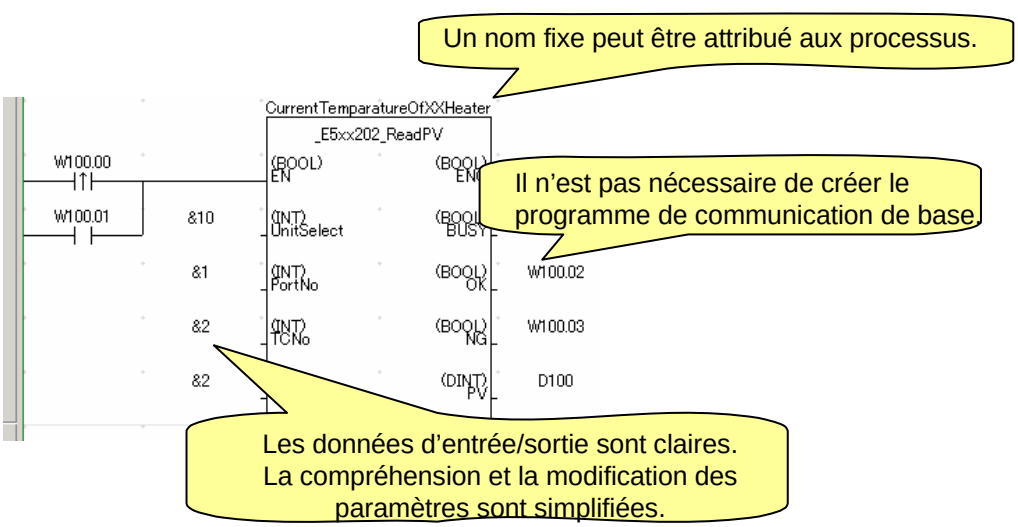
Dans le bloc de fonctions, un nom s'affiche clairement pour le corps et les instances. Un nom fixe peut également être appliqué au processus.

L'instance (instruction d'appel d'un bloc de fonctions) dispose de paramètres d'entrée et de sortie. Le relais temporaire et les données de traitement n'étant pas affichés, la visibilité des valeurs des entrées et des sorties est améliorée. En outre, comme la modification des paramètres est localisée, il est plus simple de réaliser une commande précise lors du débogage.

Enfin, le traitement interne du bloc de fonctions ne s'affiche pas en cas d'utilisation de l'instance dans le schéma de contact. Par conséquent, l'aspect du programme est simplifié pour l'utilisateur.

(5) Evolutivité ultérieure

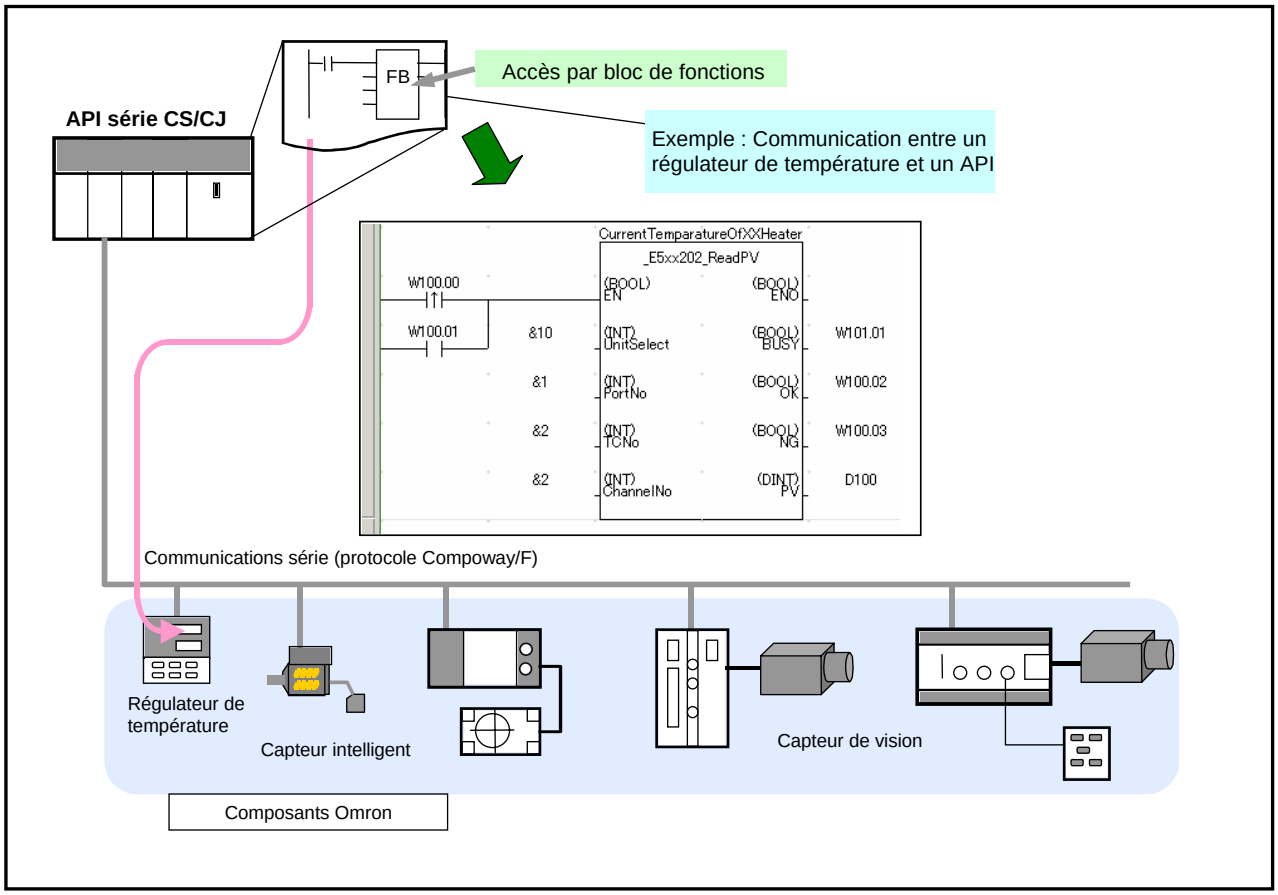
Omron ne modifiera pas l'interface entre le schéma de contact et les blocs de fonctions. A des fins d'amélioration des performances, les unités continueront à fonctionner en remplaçant le bloc de fonctions par le bloc correspondant pour la nouvelle unité, en cas de mise à niveau de l'API et des composants FA.



3-2-1. Exemple d'utilisation de la bibliothèque FB OMRON - 1

La commande des composants prédéfinis fabriqués par Omron est simplifiée à partir du schéma de contact de l'API.

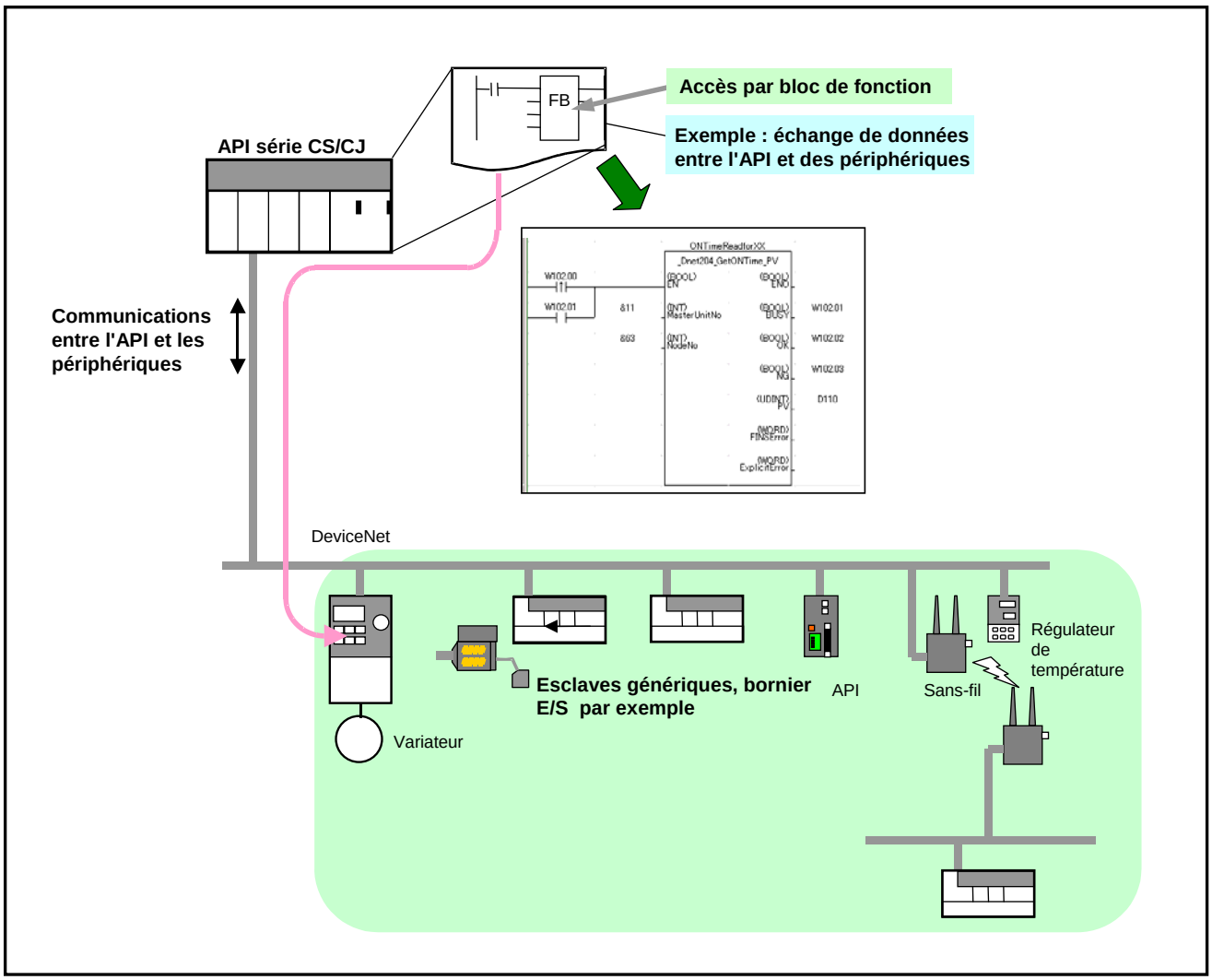
- Possibilité de configurer des communications économiques (RS-232C/485)



3-2-2. Exemple d'utilisation de la bibliothèque FB OMRON - 2

Il est possible d'obtenir des communications hautes performances grâce au niveau DeviceNet.

- Possibilité de communiquer facilement entre un API et des esclaves DeviceNet.

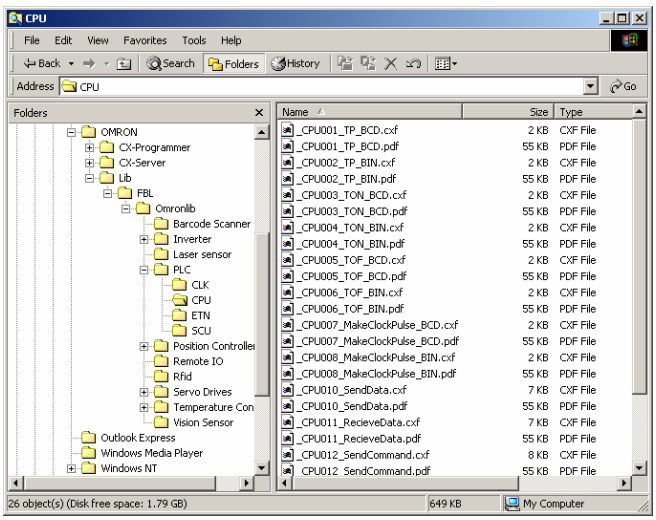


3-3. Contenu de la bibliothèque FB OMRON

La bibliothèque FB OMRON est constituée des éléments suivants :

3-3-1. Fichier de définition FB OMRON

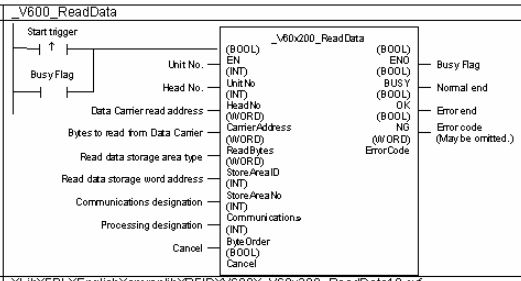
Le fichier de définition FB OMRON est préparé à l'aide du bloc de fonctions du schéma de contact afin de définir chaque fonction de l'API et du composant FA.
Le fichier contient un programme écrit dans un schéma de contact et possède l'extension .CXF.
Le nom du fichier de définition FB OMRON commence par un trait de soulignement (« _ »).
Lors de l'installation de la bibliothèque FB OMRON sur un ordinateur, les fichiers de partie FB OMRON sont classés dans le dossier correspondant à chaque API et composant dans le répertoire d'installation Omron.



3-3-2. Référence de bibliothèque

La référence de bibliothèque décrit les caractéristiques de fonctionnement du fichier de définition FB OMRON, ainsi que des paramètres d'entrée et de sortie. Il s'agit d'un fichier au format PDF.
Lors de l'utilisation de la bibliothèque FB OMRON, l'utilisateur doit sélectionner le fichier de définition FB OMRON, définir les paramètres d'entrée et de sortie et tester le fonctionnement du programme par rapport à la référence de bibliothèque.

V60x 200	Read Data Carrier Data _V60x200_ReadData

FB name	_V600_ReadData
Symbol	
File name	¥Lib¥FBL¥English¥omronlib¥RFID¥V600¥_V60x200_ReadData10.cxf
Applicable models	CS1W-V600C11/V600C12 and CJ1W-V600C11/V600C12 ID Sensor Units
Basic function	Reads data from a Data Carrier.
Conditions for usage	Other • This FB cannot be executed if the ID Sensor Unit is busy. The NG Flag will turn ON if an attempt is made.
Function description	Data is read from the specified area of the Data Carrier specified by the *Unit No.* and *Vendor No.* Up to 2048 bytes (1024 words) can be read at one time. The word designation for storing the data is specified using the area type and beginning word address. For example, for D1000, the area type is set to P_DM and the beginning word address is set to &1000.
EN input condition	Connect EN to an OR between an upwardly differentiated condition for the start trigger and the BUSY output from the FB.
Restrictions Input variables	• Always use an upwardly differentiated condition for EN. • If the input variables are out of range, the ENO Flag will turn OFF and the FB will not be processed. • Always specify a head number of &1 for One-Head ID Sensor Units (CS1W-V600C11 and CJ1W-V600C11).

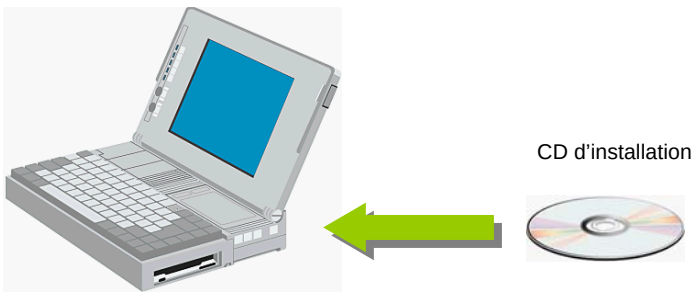
3-4. Catalogue de fichiers et accès à la bibliothèque FB OMRON

3-4-1. Catalogue des fichiers de la bibliothèque FB OMRON

Type	Composants cibles	Nombre de fichiers de partie FB OMRON (février 2005)
Composants	Régulateur de température, capteur intelligent, capteur ID, capteur de vision, lecteur de codes-barres à 2 dimensions, terminal sans fil	environ 80
API	UC, carte mémoire, cartes E/S spéciale (cartes Ethernet, ControllerLink, DeviceNet, régulation de température)	environ 95
Composants de contrôle d'axes	Carte de contrôle de position Variateur Servodriver	environ 70

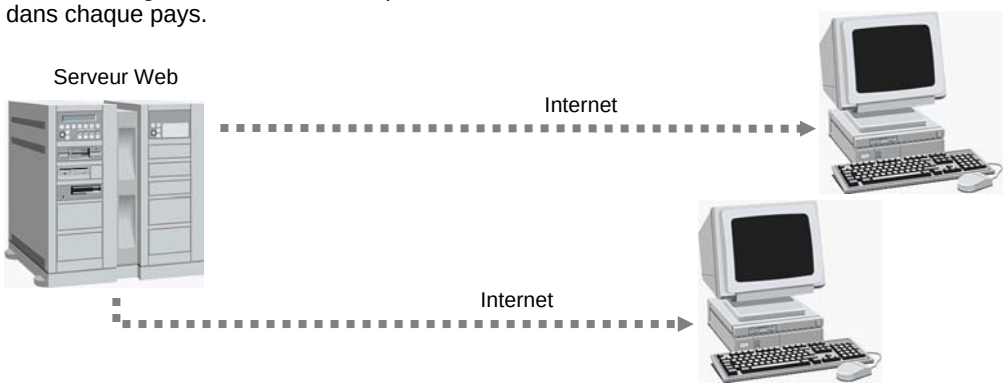
3-4-2. CD-ROM d'installation de CX-One / CX-Programmer

La bibliothèque FB OMRON figure sur le même CD d'installation que CX-One / CX-Programmer.
Son installation peut être sélectionnée lors de l'installation de CX-One / CX-Programmer.



3-4-3. Accès aux fichiers de la bibliothèque FB OMRON à partir d'un serveur Web

La dernière version des fichiers de la bibliothèque FB OMRON est fournie par Omron sur le serveur Web.
De nouveaux fichiers seront ajoutés afin d'assurer la prise en charge des API et composants nouveaux ou améliorés.
Le service de téléchargement de la bibliothèque FB OMRON est accessible dans un menu sur le site Web d'Omron dans chaque pays.



Chapitre 2

Utilisation de la bibliothèque FB OMRON

Function Block

1. Explication du programme cible

Ce chapitre explique comment utiliser la bibliothèque FB OMRON à l'aide du fichier de définition FB OMRON « Make ON Time/OFF Time Clock Pulse in BCD ».

1-1. Caractéristiques de l'application

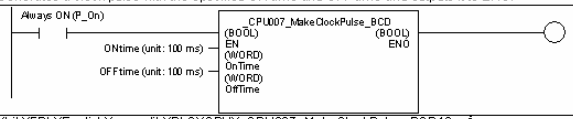
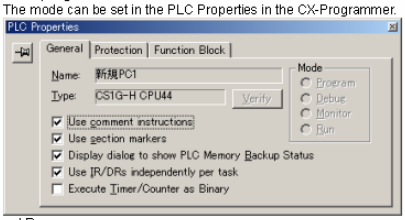
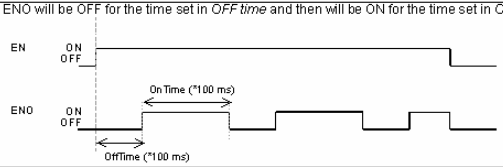
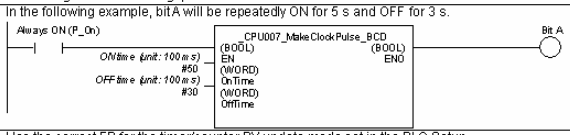
Les caractéristiques de l'application cible sont les suivantes :

- L'impulsion est générée lorsque l'API passe en mode d'exécution (run) ou de surveillance (monitor).
- Sortie de l'impulsion vers l'adresse 1.00.
- Le temps d'activité de l'impulsion générée est défini sur D100.
- Le temps d'inactivité de l'impulsion générée est de 2 secondes.

1-2. Caractéristiques du fichier de définition FB OMRON

Le fichier de définition FB OMRON « Make ON Time/OFF Time Clock Pulse in BCD » présente les caractéristiques suivantes :

CPU 007	Make ON Time/OFF Time Clock Pulse in BCD _CPU007_MakeClockPulse_BCD

Basic function	Generates a clock pulse with the specified ON time and OFF time and outputs it to ENO.
Symbol	
File name	¥Lib¥FBL¥English¥omronlib¥PLC¥CPU¥_CPU007_MakeClockPulse_BCD10.cxf
Applicable models	CS1-H, CS1-H, and CJ1M CPU Units
Conditions for usage	PLC Properties • The PV update method for timers and counters must be set to BCD in the PLC Setup. A compiling error will occur if BCD mode is not set. The mode can be set in the PLC Properties in the CX-Programmer.  Shared Resources • Timers
Function description	ENO will be OFF for the time set in *OFF time* and then will be ON for the time set in *ON time*. 
EN input condition	Connect the EN input to the Always ON Flag (P_On).
Restrictions Input variables	• If the input variables are out of range, the ENO Flag will turn OFF and the FB will not be processed. • Set the *ON time* and *OFF time* input variables to between #0000 and #9999 in BCD (100 ms units). If a setting is not within range, ENO is turned OFF.
Application example	In the following example, bit A will be repeatedly ON for 5 s and OFF for 3 s. 
Related FBs	Use the correct FB for the timer/counter PV update mode set in the PLC Setup. Binary mode: Make ON Time/OFF Time Clock Pulse in Binary (_CPU008_MakeClockPulse_BIN) BCD mode: Make ON Time/OFF Time Clock Pulse in BCD (_CPU007_MakeClockPulse_BCD)

■ Variable Tables

Input Variables

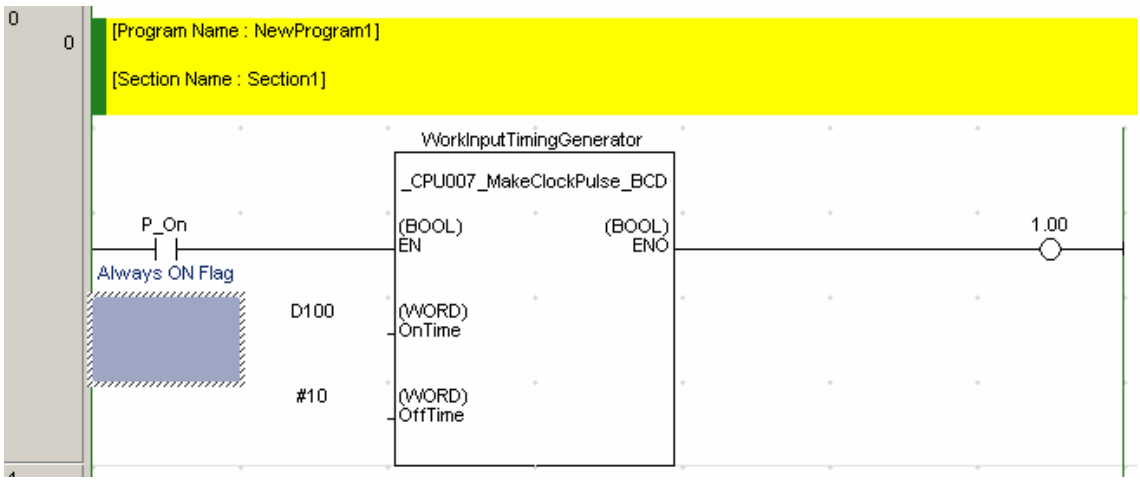
Name	Variable name	Data type	Default	Range	Description
EN	EN	BOOL			1 (ON): FB started 0 (OFF): FB not started.
ON time	OnTime	WORD		#0000 to #9999	Specify the ON time (unit: 100 ms). For example, #30 means 3 seconds.
OFF time	OffTime	WORD		#0000 to #9999	Specify the OFF time (unit: 100 ms). For example, #30 means 3 seconds.

Output Variables

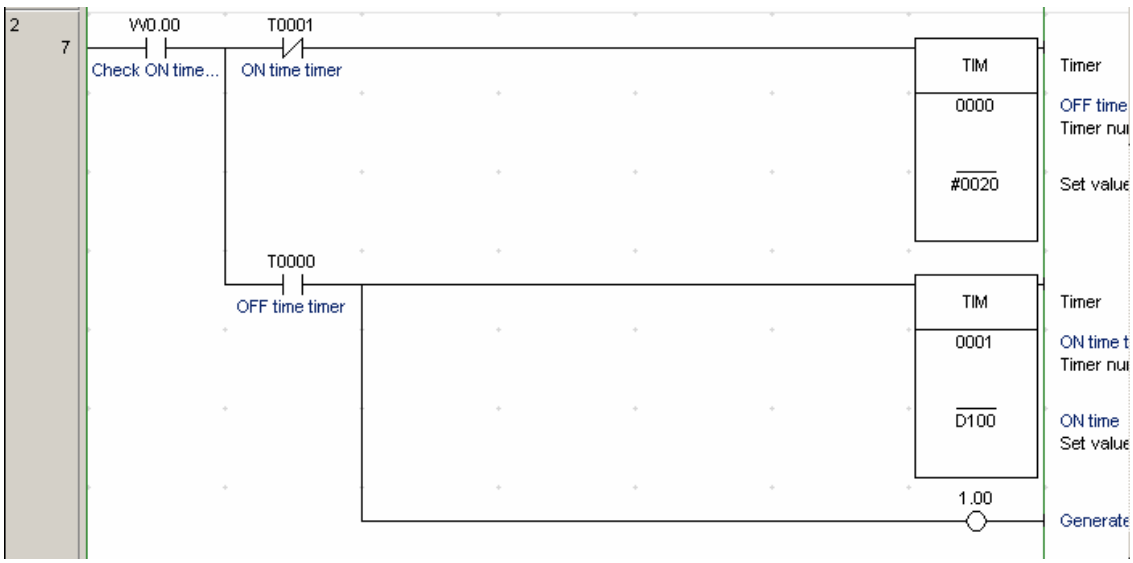
Name	Variable name	Data type	Range	Description
ENO	ENO	BOOL		Turns ON for the OnTime and OFF for the OffTime.

1-3. Programme d'entrée

Créez le programme suivant :



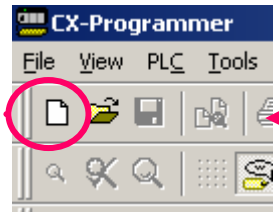
[Référence] S'il est créé en tant que schéma de contact direct, le programme se présente comme suit :

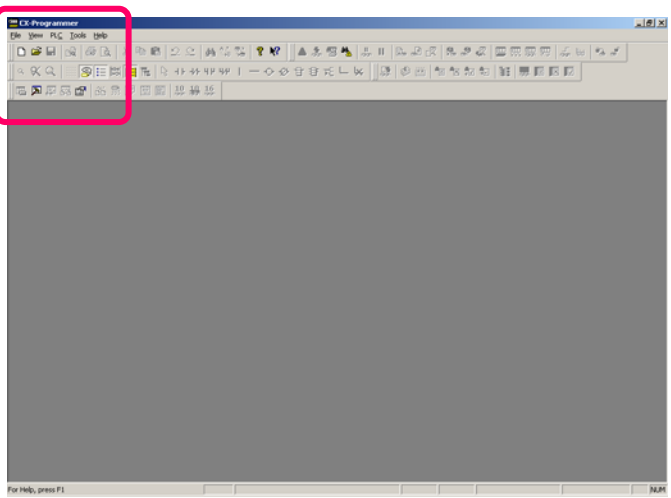


2. Ouverture d'un nouveau projet et configuration du type d'appareil

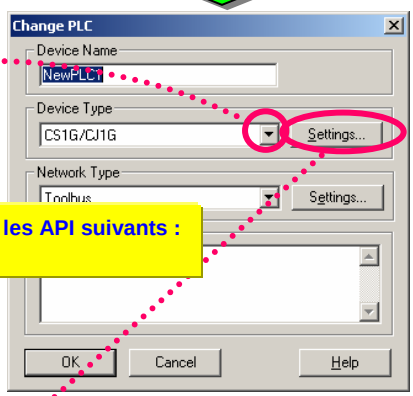
Cliquez sur le bouton de barre d'outils [Nouveau] dans CX-Programmer.

Cliquez sur 



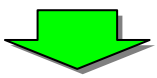


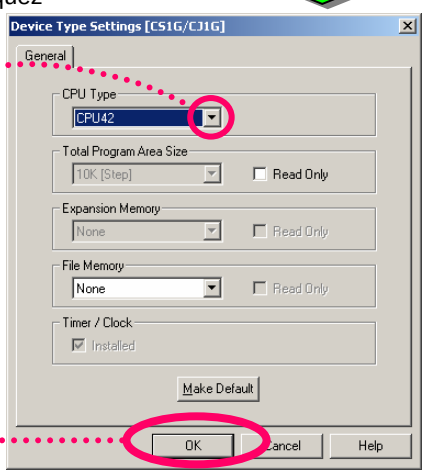
Cliquez sur 



! Pour utiliser les blocs de fonctions, sélectionnez les API suivants :
CS1G-H, CS1H-H, CJ1G-H, CJ1H-H, CJ1M

Cliquez 



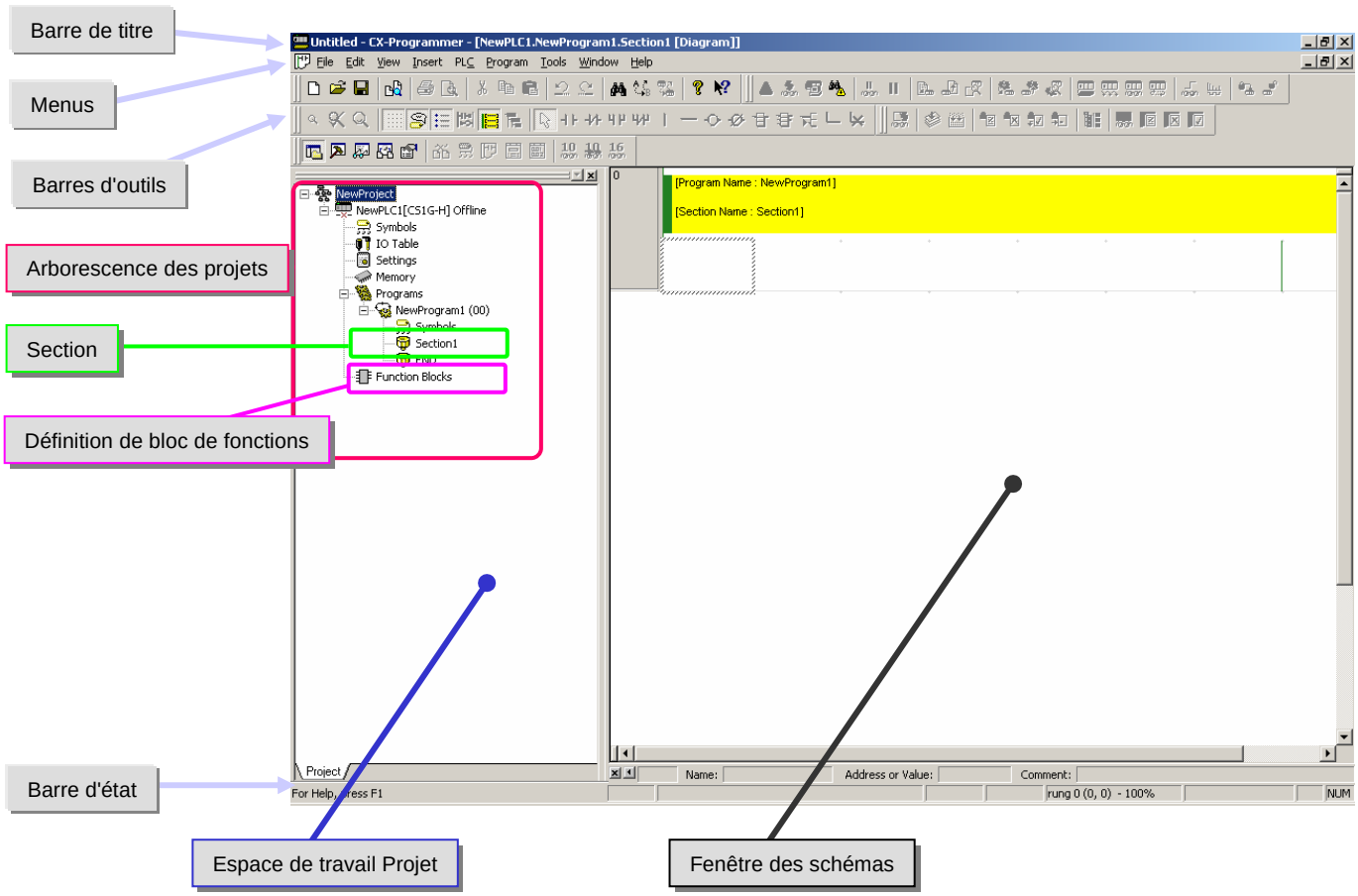


Cliquez pour sélectionner le type de carte UC. 

Cliquez sur [OK] pour valider la sélection de la carte UC.

3. Fonctions de la fenêtre principale

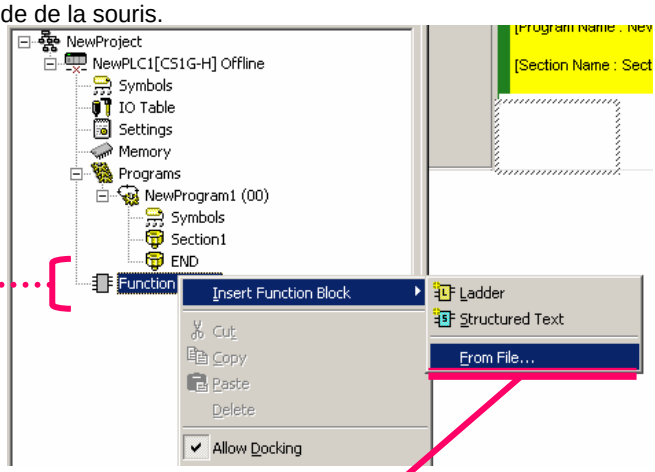
Vous trouverez ci-dessous l'explication des fonctions de la fenêtre principale.



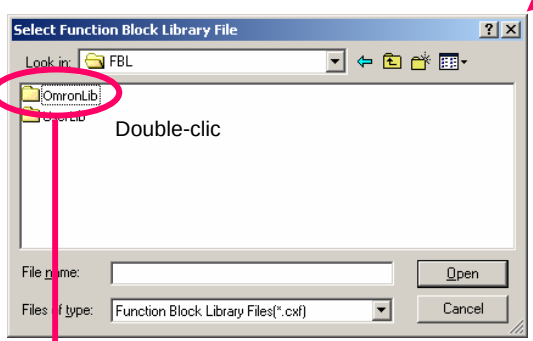
Nom	Contenu/Fonction
Barre de titre	Affiche le nom du fichier des données enregistrées et créées dans CX-Programmer.
Menus	Permettent de sélectionner des options de menu.
Barres d'outils	Permettent de sélectionner des fonctions en cliquant sur des icônes. Sélectionnez [Affichage] -> [Barres d'outils] pour afficher les barres d'outils. Déplacez des barres d'outils pour modifier la position d'affichage.
Section	Permet de diviser un programme en plusieurs blocs. Il est possible de créer et d'afficher séparément chacun d'entre eux.
Espace de travail Projet Arborescence des projets	Détermine les programmes et les données. Permet de copier des données par glisser-déplacer entre deux projets ou à l'intérieur d'un même projet.
Fenêtre des schémas	Ecran permettant de créer et de modifier un programme schéma contcts.
Définition de bloc de fonctions	Affiche la définition de bloc de fonctions. En cliquant sur les icônes, vous pouvez copier ou supprimer la définition de bloc de fonctions sélectionnée. - s'affiche si le fichier est un fichier de définition FB OMRON. - Dans le cas d'un bloc de fonctions défini par l'utilisateur, s'affiche pour un schéma ou pour du texte structuré.
Barre d'état	Affiche des informations telles que le nom de l'API, l'état en ligne/hors ligne, l'emplacement de la cellule active.

4. Importation du fichier de définition FB OMRON

Cliquez sur l'icône correspondant à la définition du bloc de fonctions dans l'arborescence du projet, puis cliquez avec le bouton droit. Sélectionnez Insérer bloc fonction, puis sélectionnez un fichier de bibliothèque à l'aide de la souris.



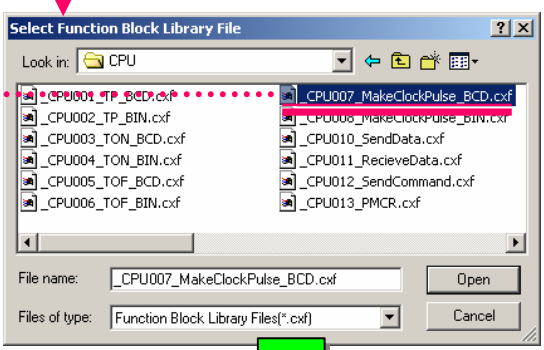
Cliquez avec le bouton droit.
→ Insérer bloc fonction
→ Fichier de bibliothèque



Double-cliquez.
→ [OmronLib]
→ [Programmable Controller]
→ [CPU]
Sélectionnez chaque élément ci-dessus dans l'ordre.

Sélectionnez le fichier de définition FB OMRON de votre choix dans la boîte de dialogue Sélection d'un fichier Librairie de bloc fonction.

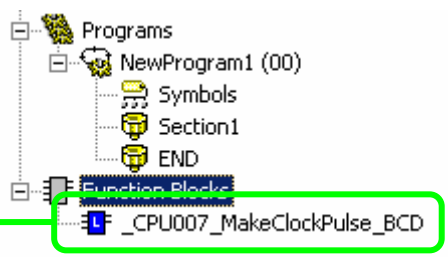
! Le chemin par défaut de la bibliothèque FB OMRON est C:\Program Files\Omron\CX-One\Lib\FBL.



Cliquez sur « _CPU007_MakeClockPulse_BCD.cxf »

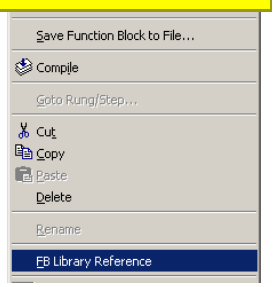
Cliquez sur le bouton [Ouvrir].

La définition de bloc de fonctions « _CPU007_MakeClockPulse_BCD » est enregistrée dans le fichier de projet.



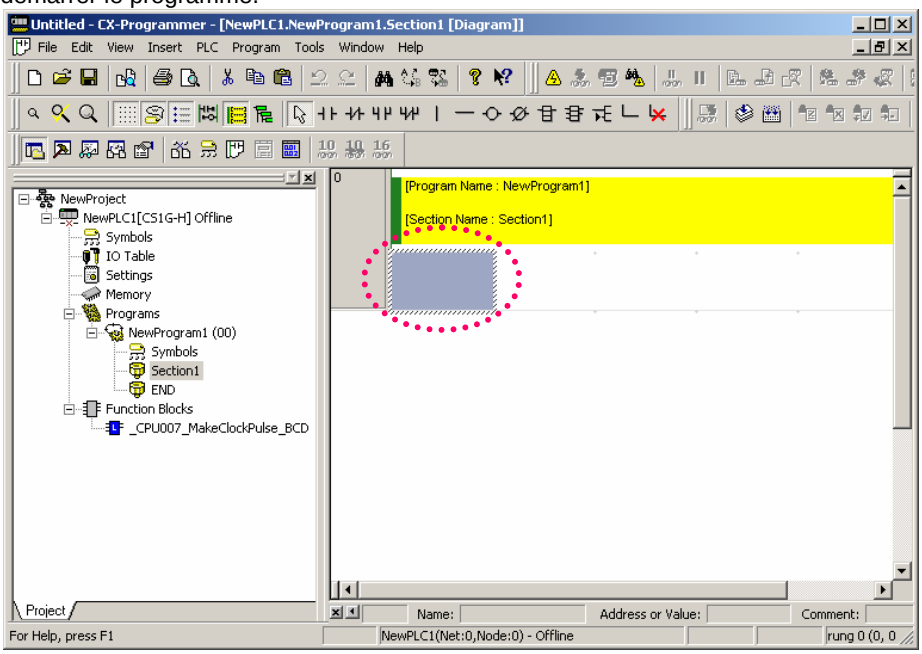
Définition de bloc de fonctions

! Pour consulter rapidement les caractéristiques d'un fichier de définition FB OMRON, sélectionnez le fichier enregistré, puis sélectionnez [Référence bibliothèque FB] dans le menu contextuel. Le fichier de référence de la bibliothèque s'affiche alors.



5. Création d'un programme

Assurez-vous que le curseur se trouve en haut à gauche dans la fenêtre Schémas avant de démarrer le programme.



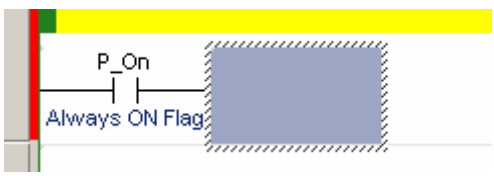
5-1. Entrée d'un contact normalement ouvert

C

P_On

ENT

Appuyez sur la touche [C] du clavier pour ouvrir la boîte de dialogue [Nouveau contact].
Sélectionnez le symbole P_On dans la liste déroulante.



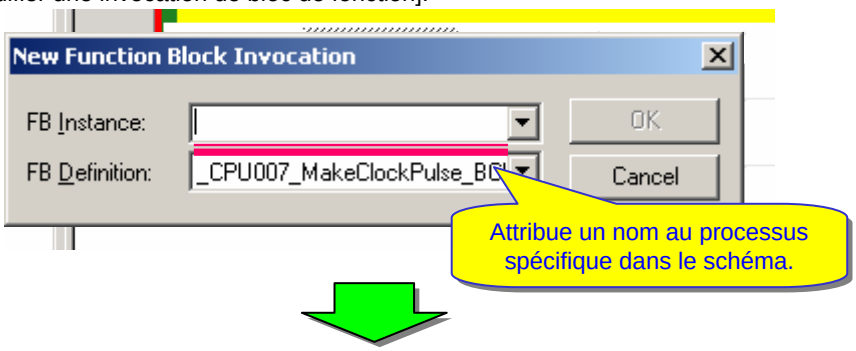
Suppression d'une commande
- Placez le curseur sur la commande, puis appuyez sur la touche Suppr ou
- Placez le curseur sur la cellule à droite de la commande, puis appuyez sur la touche Retour arrière.

P_On est un symbole défini par le système. Il a toujours l'état ON (activé).
0 ne s'affiche pas lorsqu'il est le premier chiffre d'une adresse.
Le point [.] sépare le numéro de canal et le numéro de relais.

5-2. Entrée d'une instance

F

Appuyez sur la touche [F] du clavier pour ouvrir la boîte de dialogue [Modifier une invocation de bloc de fonction].

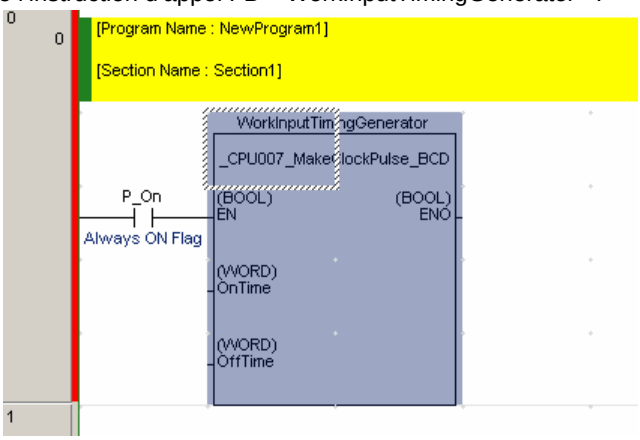


Entrez du texte pour attribuer un nom à l'instance FB.

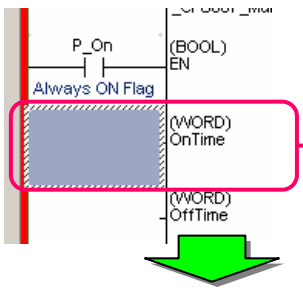
[WorkInputTimingGenerator]

ENT

Affiche l'instruction d'appel FB « WorkInputTimingGenerator ».



5-3. Entrée de paramètres ou ENT



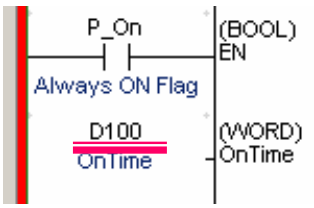
Placez le curseur à gauche du paramètre d'entrée.

P

Entrez l'adresse.

[d100]

ENT



Sélectionnez l'adresse du paramètre d'entrée OnTime.

Entrez les autres paramètres en suivant la même procédure.

P ou ENT

#10

ENT

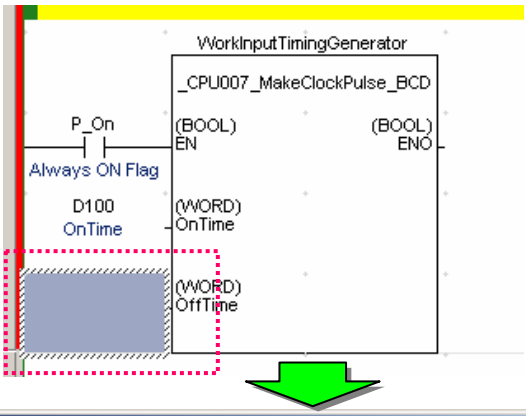
O

1.00

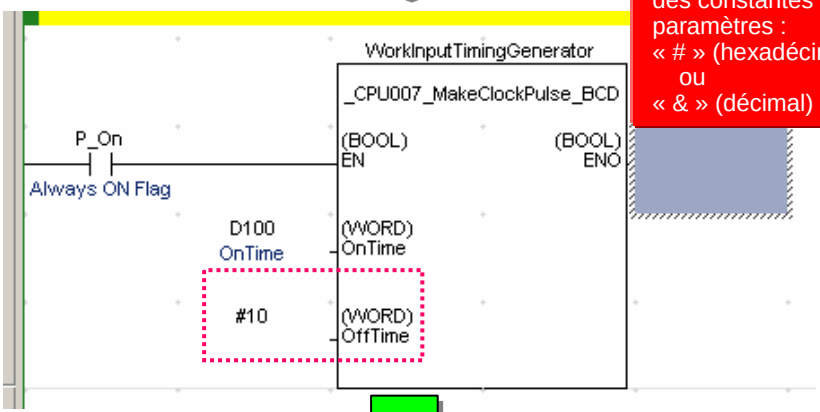
ENT

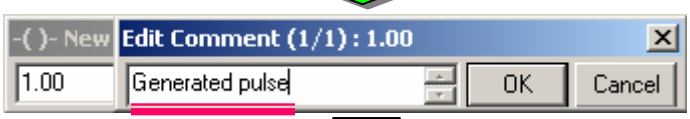
[Impulsion générée]

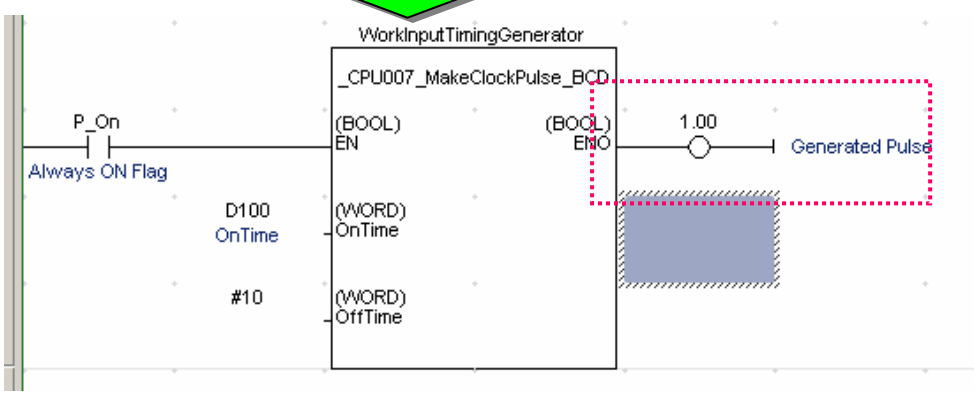
ENT



Ajoutez le préfixe suivant pour entrer des constantes en tant que paramètres :
« # » (hexadécimal/BCD)
ou
« & » (décimal)



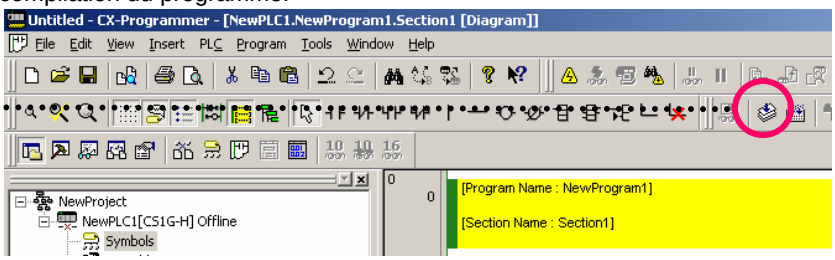


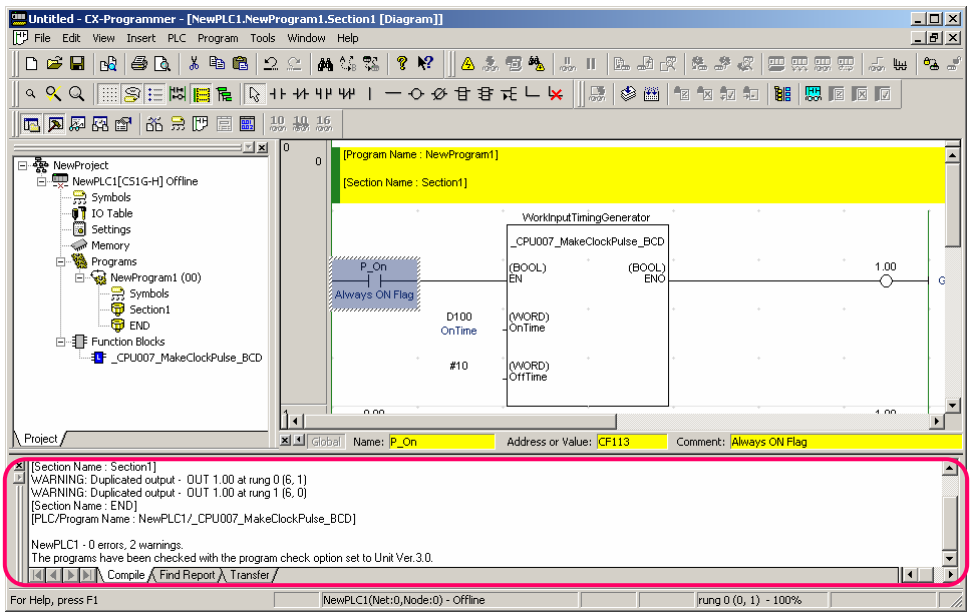


6. Contrôle d'erreurs de programme (compilation)

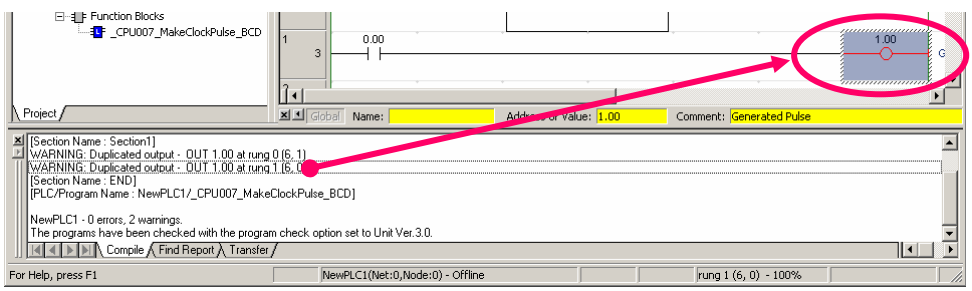
Avant de transférer le programme, contrôlez les erreurs éventuelles à l'aide de la fonction de compilation du programme.

Cliquez sur 





Les erreurs et les adresses correspondantes s'affichent dans la fenêtre Sortie.



Double-cliquez sur une erreur affichée ; le curseur du schéma de contact se déplace alors vers l'erreur correspondante. Le segment en erreur s'affiche en rouge.

Corrigez l'erreur.

- La fenêtre Sortie s'ouvre automatiquement lors du contrôle du programme.
- Pour déplacer le curseur sur une erreur, appuyez sur la touche J ou F4.
- Pour fermer la fenêtre Sortie, appuyez sur la touche Echap.

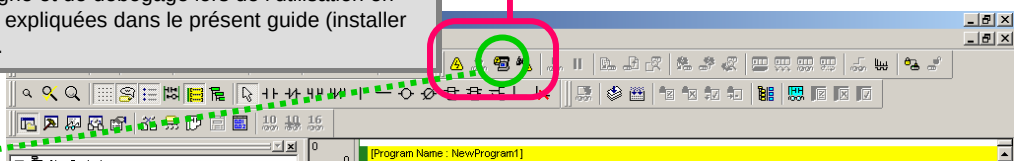
7. Mise en ligne

CX-Programmer propose trois modes de connexion qui varient en fonction de l'application.

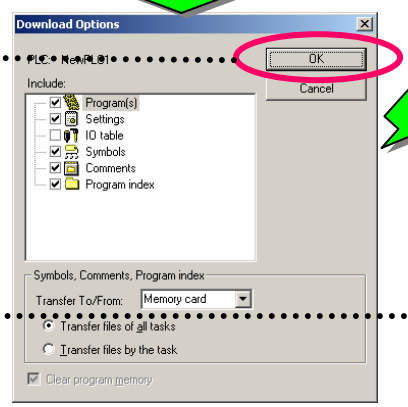
- Online normal. Permet d'activer le mode en ligne pour un API de type et avec la méthode indiqués lors de l'ouverture d'un projet.
- Mode online automatique. Ce mode reconnaît l'API connecté et vous permet d'activer le mode en ligne de l'API d'un seul bouton.
  → Permet de télécharger toutes les données, comme les programmes, à partir de l'API.
- Online avec simulateur. Permet d'activer le mode en ligne de CX-Simulator d'un seul bouton (CX-Simulator doit être installé).

Les fonctions de mise en ligne et de débogage lors de l'utilisation en ligne de CX-Simulator sont expliquées dans le présent guide (installer CX-Simulator séparément).

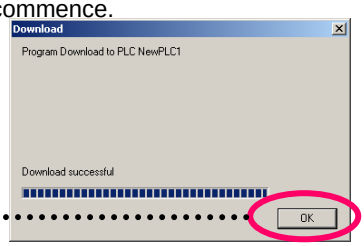
Cliquez sur 



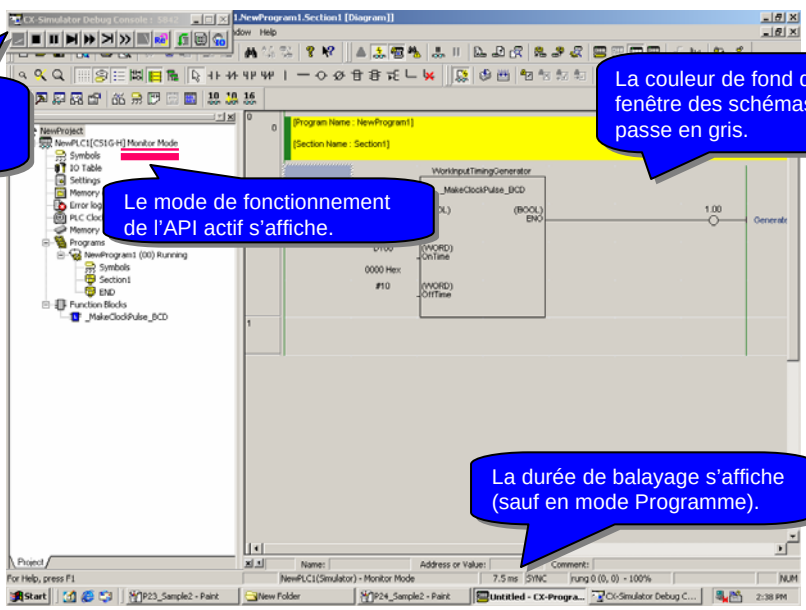
Cliquez sur [OK]



Le transfert du programme commence.



Cliquez sur [OK]

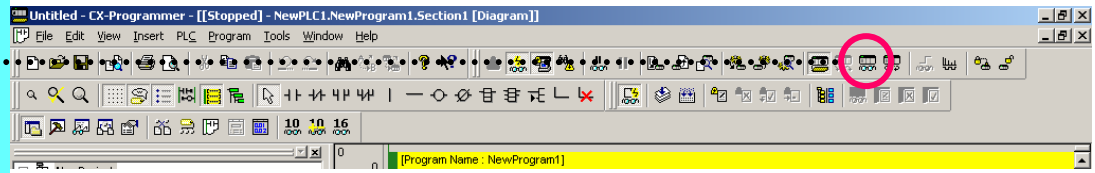


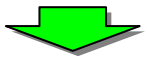
8. Surveillance - 1

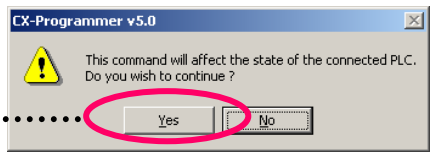
Faites passer l'API (simulateur) en mode de surveillance.

Il est possible de surveiller l'état on/off des contacts et des bobines.

Cliquez sur 

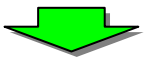




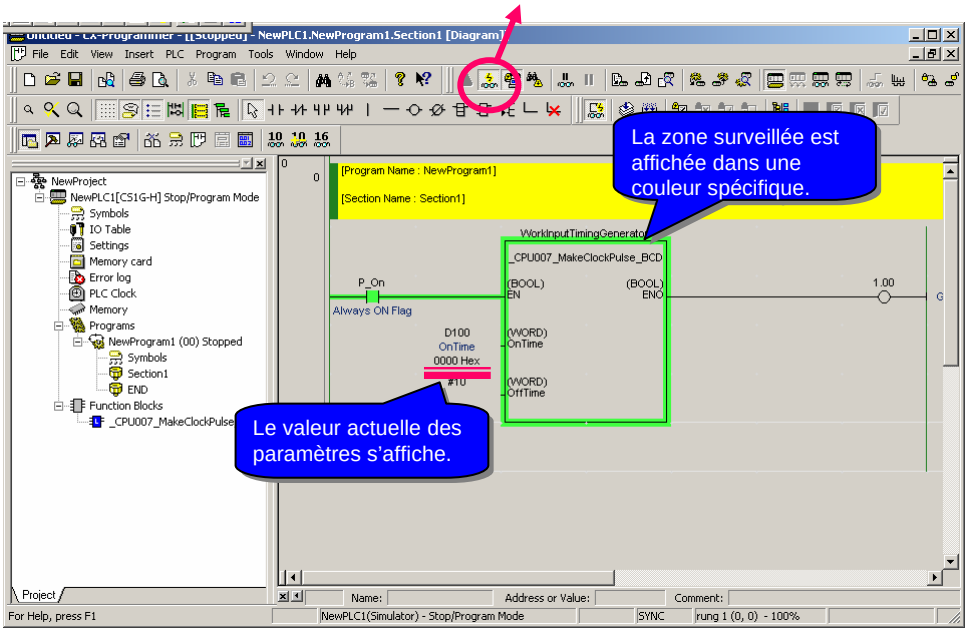


Cliquez sur [Oui].

Il est possible que la vitesse de défilement de l'écran soit ralentie lors de la surveillance si le programme comporte un volume important de données.
Pour résoudre ce problème, cliquez sur l'icône ci-dessous pour annuler la surveillance, faites défiler l'écran pour afficher l'adresse à placer sous surveillance, puis redémarrez le mode de surveillance.



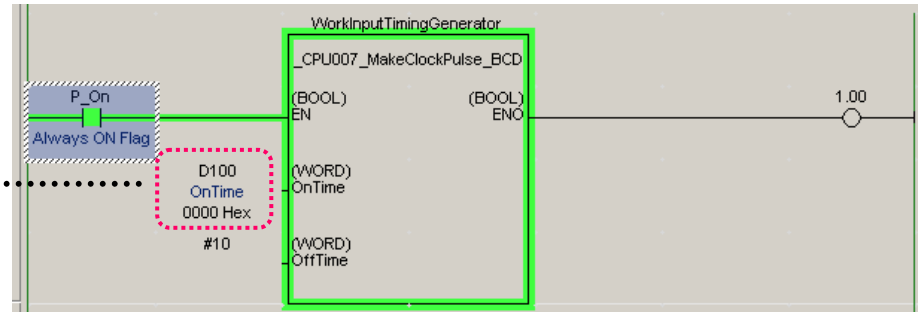
 ••• active/désactive la surveillance d'un API



9. Surveillance - 2 Modification de la valeur actuelle d'un paramètre

Modifiez la valeur actuelle d'un contact/bobine ou des données de mot dans la fenêtre Schéma.

Placez le curseur sur le paramètre d'entrée D100.



Cliquez avec le bouton droit, puis sélectionnez l'option de menu

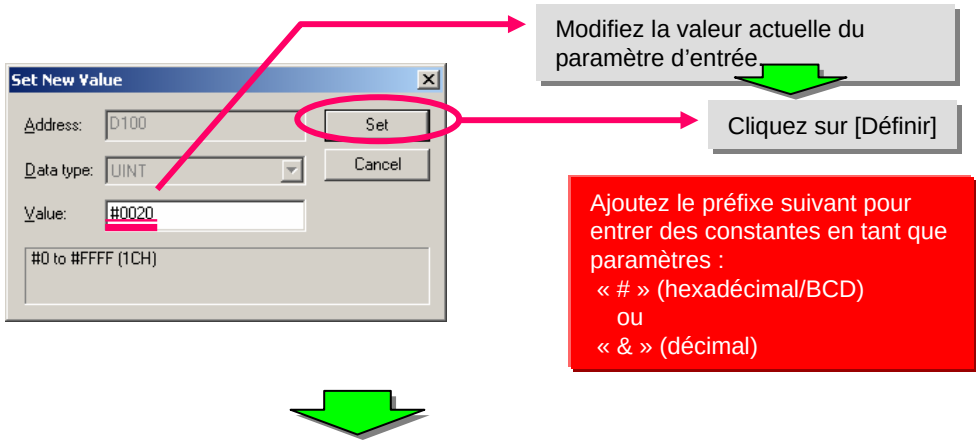
[Initialiser/Réinitialiser]
→ [Valeur de configuration]

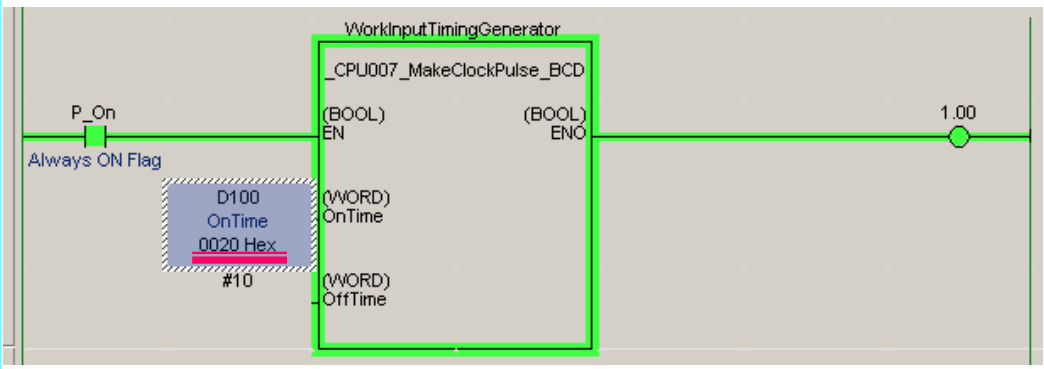
ou

Double-cliquez.

Ou

ENT

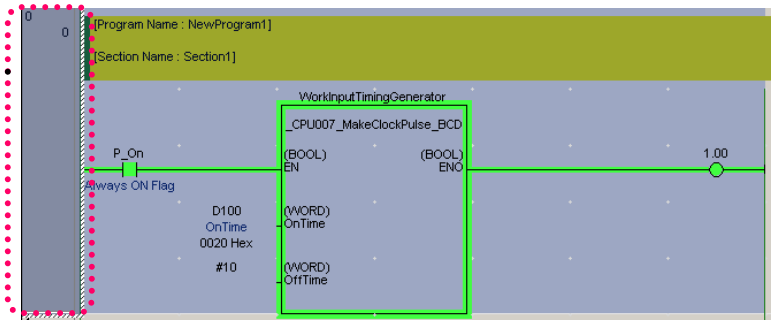




10. Edition en ligne

Placez le curseur sur le segment à modifier.

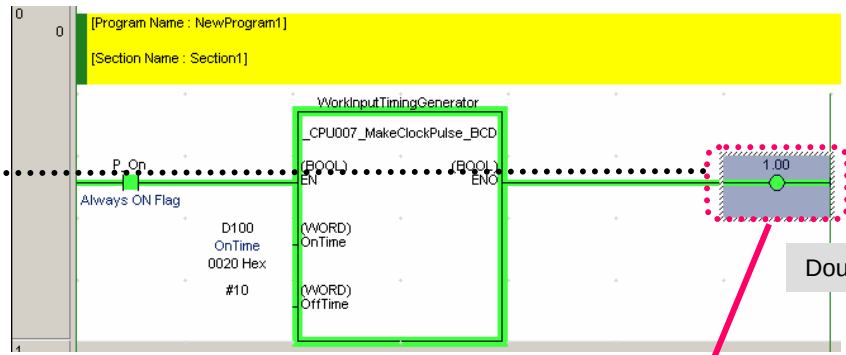
Vous avez la possibilité de sélectionner plusieurs segments en utilisant la fonction Glisser/Déplacer (avec la souris).



Sélectionnez [Programme]
→ [Edition en ligne]
→ [Commencer]

Raccourci : [Ctrl]+[E]

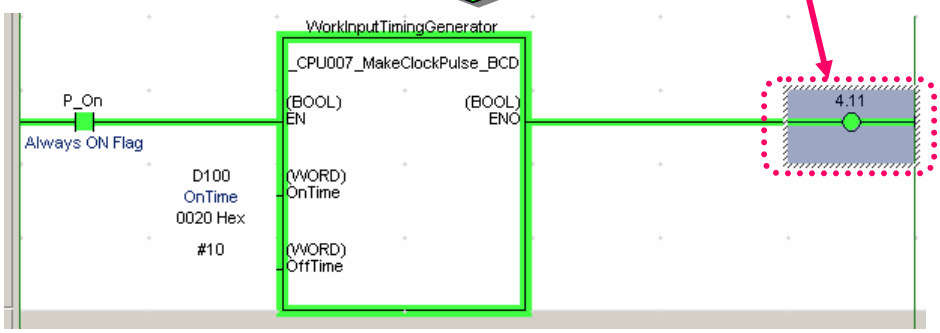
Placez le curseur sur la bobine à modifier.
Double-cliquez.



Modifiez l'adresse en fonction du numéro de bit requis (4.11 dans cet exemple).

Sélectionnez [Programme]
→ [Edition en ligne]
→ [Envoyer Changements]

Raccourci : [Ctrl]+[Maj]+[E]



Fin

Chapitre 3

Personnalisation du fichier de définition FB OMRON

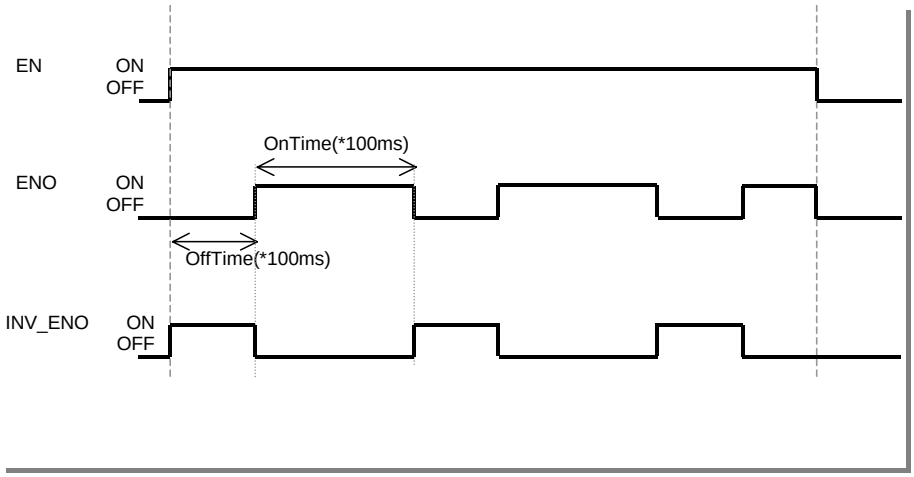
Function Block

1. Explication du programme cible

Ce chapitre explique comment personnaliser la bibliothèque FB OMRON à l'aide du fichier de définition FB OMRON « Make ON Time/OFF Time Clock Pulse in BCD ».

1-1. Modification des caractéristiques du fichier

Le fichier de définition FB OMRON « Make ON Time/OFF Time Clock Pulse in BCD » permet de désactiver la variable ENO de manière répétée pour la durée d'inactivité spécifiée (unité : 100 ms) et de l'activer pour la durée d'activité spécifiée (unité : 100 ms). Dans cet exemple, le fichier de définition FB OMRON sera modifié pour sortir un signal inversé en ajoutant le paramètre de sortie INV_ENO.



1-2. Modification du contenu du fichier de définition FB OMRON

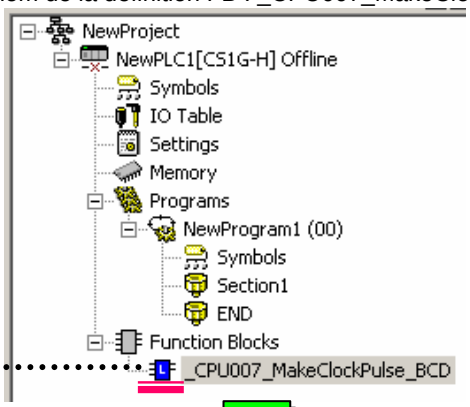
Pour répondre à la condition décrite ci-dessus, les modifications suivantes doivent être apportées au fichier de définition FB OMRON « Make ON Time/OFF Time Clock Pulse in BCD ».

1. Ajouter le paramètre de sortie INV_ENO.
2. Ajouter un programme de schéma pour sortir la variable ENO afin d'inverser le signal.

Attention
OMRON ne garantit pas le bon fonctionnement d'un fichier FB OMRON personnalisé.
Assurez-vous de vérifier le processus de la partie FB avant de la personnaliser, puis veillez à ce que chaque partie FB fonctionne correctement.

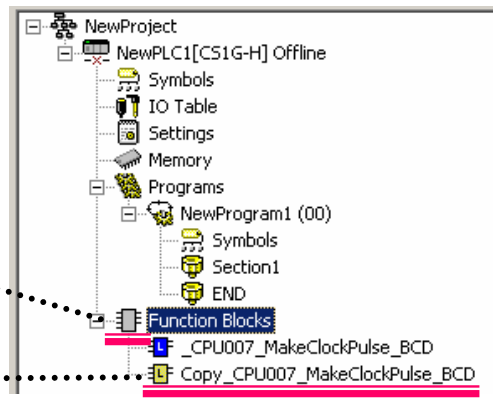
2. Copie du fichier de définition FB OMRON

Importez le fichier de définition FB OMRON « Make ON Time/OFF Time Clock Pulse in BCD » en suivant les instructions fournies au chapitre 1
(nom de la définition FB : _CPU007_MakeClockPulse_BCD).



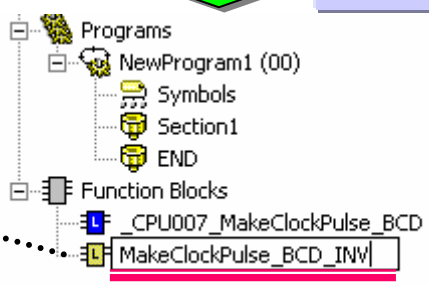
Sélectionnez l'icône de la définition FB OMRON, puis cliquez dessus avec le bouton droit.

→ Copier



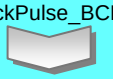
Cliquez sur l'icône de la définition du bloc de fonctions, puis cliquez avec le bouton droit.
→ Coller
Le fichier de définition FB OMRON est collé.

Modifiez le nom de la définition FB.



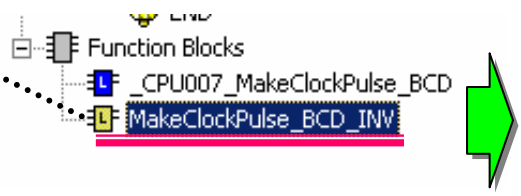
Sélectionnez l'icône du bloc de fonctions collé, puis cliquez dessus avec le bouton droit.

→ Renommer [MakeClockPulse_BCD_INV]



Remarque :
Vous ne pouvez pas créer une définition de bloc de fonctions dont le nom commence par trait de soulignement (« _ »).
N'ajoutez pas « _ » au début du nom.

Activez la modification du code programme FB interne.



Cochez la case en cliquant dessus.

Sélectionnez l'icône du bloc de fonctions collé, puis cliquez dessus avec le bouton droit.
→ Propriété

Ou

ALT + ENT

3. Ajout d'une variable à un bloc de fonctions

Ouvrez l'Editeur contact bloc fonction.

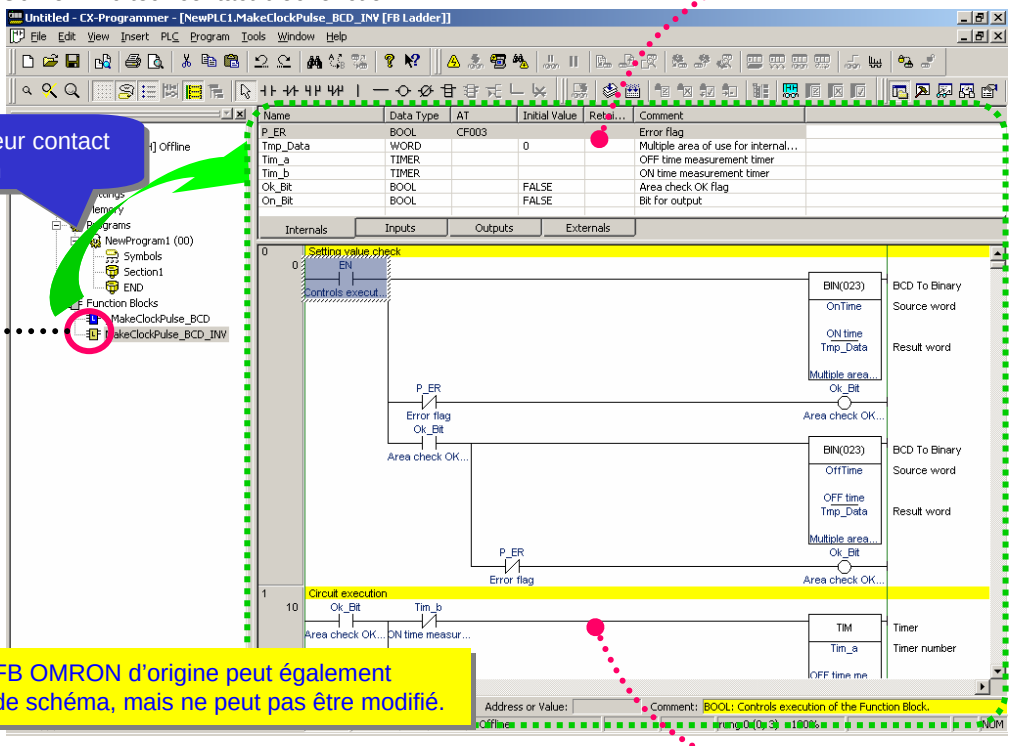


Table des variables

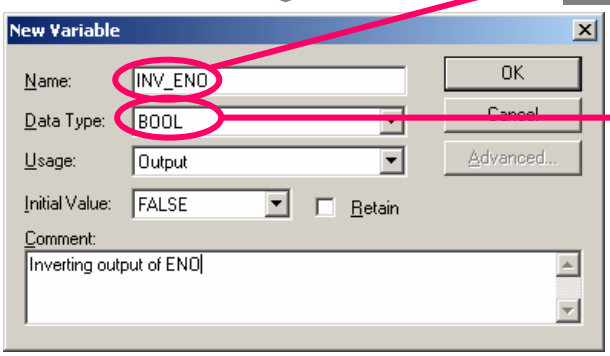
Name	Data Type	AT	Initial Value	Retai...	Comment
ENO	BOOL		FALSE		Indicates successful execution ...

Internals | Inputs | Outputs | Externals

Cliquez sur l'onglet Sorties dans la table des variables.

Cliquez, puis sélectionnez Insérer variable.

Entrez le nom de la nouvelle variable.



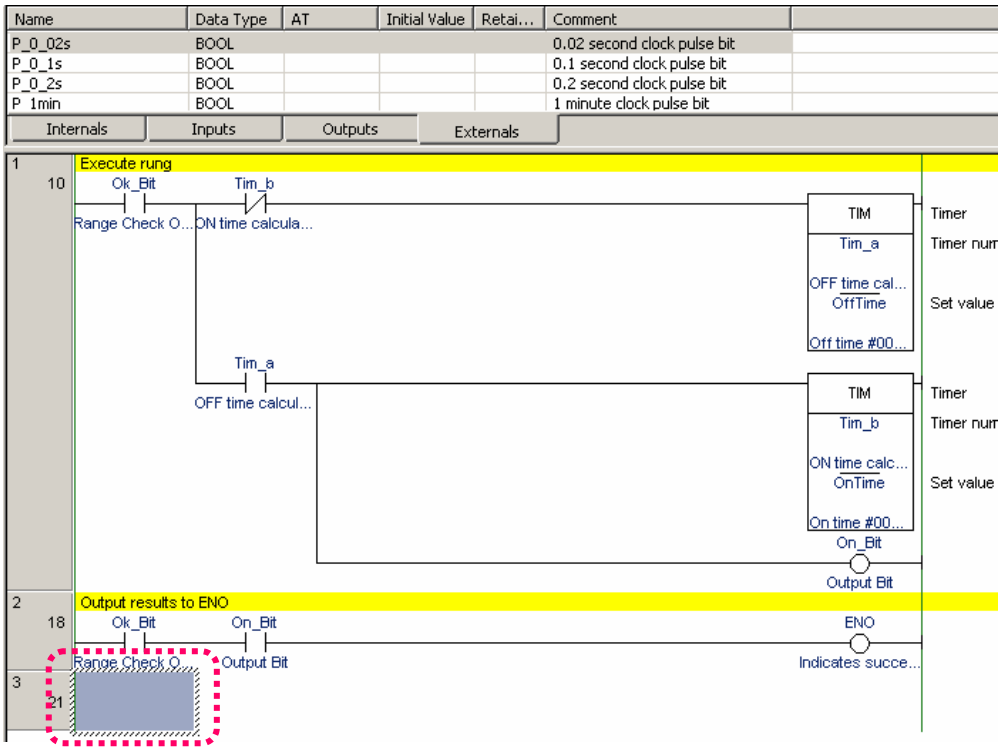
Sélectionnez BOOL comme données de bit.

Assurez-vous que la variable entrée est correcte.

Name	Data Type	AT	Initial Value	Retai...	Comment
ENO	BOOL		FALSE		Indicates successful execution ...
INV_ENO	BOOL		FALSE		Inverting output of ENO

4. Modification du schéma de blocs de fonctions

Ajoutez le schéma de contact requis dans le champ de modification du contact de bloc de fonctions. Placez le curseur dans la colonne de gauche du segment suivant.



4-1. Entrée d'un contact

/

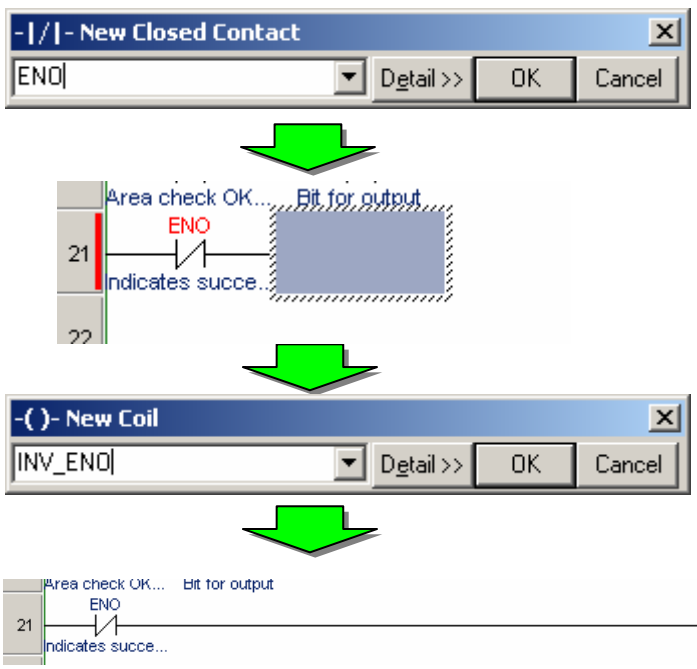
ENO

ENT

O

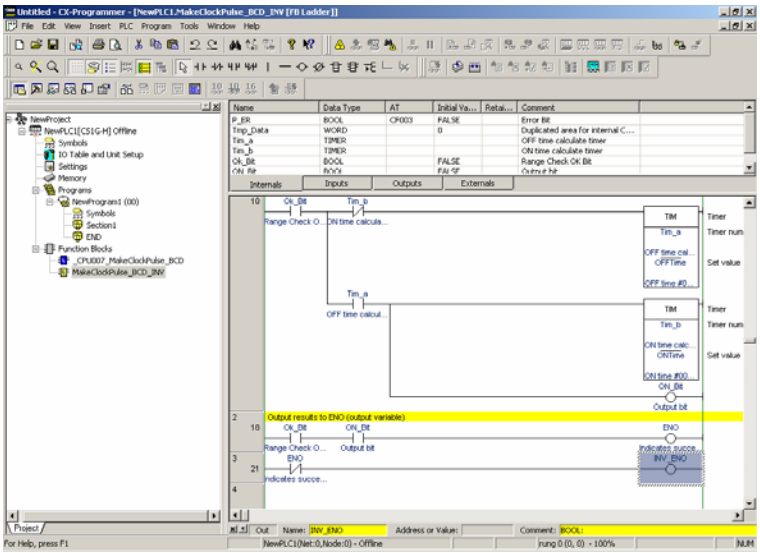
INV_ENO

ENT



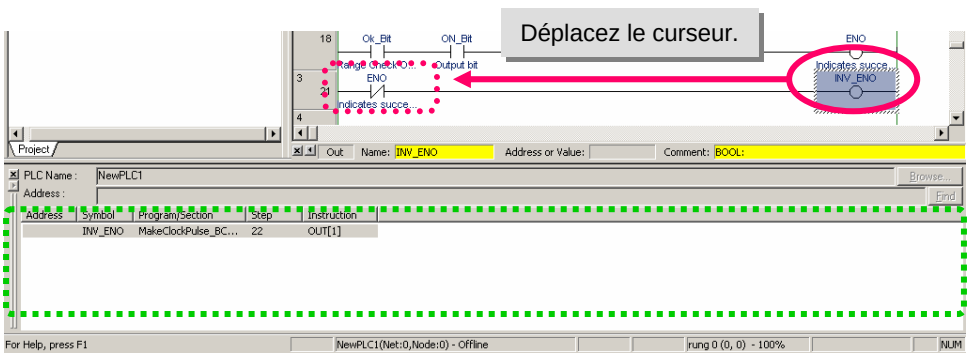
4-2. Vérification de l'état d'utilisation de variables

A l'instar du programme principal, vous pouvez utiliser le tableau de références croisées pour contrôler les conditions d'utilisation des variables.

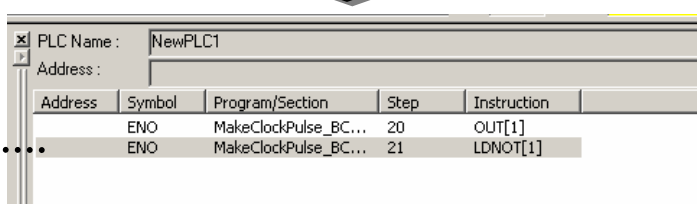


Afficher le tableau des références croisées.

Alt + 4



Sélectionnez LDNOT dans le tableau de références croisées à l'aide du curseur de la souris.



Vous pouvez constater que la variable ENO est également utilisée dans une bobine de sortie à l'étape 20.

Le curseur de l'éditeur de contact FB se place sur la bobine de sortie de l'étape 20.

Chapitre 4

Utilisation du langage texte structuré(ST)

Function Block

1. Définition du langage ST

Le langage de texte structuré (ST, Structured Text) est un code langage de haut niveau conçu pour les commandes industrielles (principalement les API) et défini par la norme IEC 61131-3.
Il contient de nombreuses instructions de commande, dont IF-THEN-ELSE-END_IF, la boucle FOR / WHILE, ainsi que de nombreuses fonctions mathématiques telles que SIN / LOG. Il convient pour le traitement mathématique.
Le langage ST pris en charge par CX-Programmer est conforme à la norme IEC 61131-3.
Les fonctions arithmétiques de CX-Programmer version 5/6 sont les suivantes :

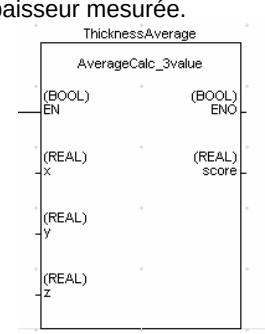
sinus (SIN), cosinus (COS), tangente (TAN), arc sinus (ASIN), arc cosinus (ACOS), arc tangente (ATAN), racine carrée (SQRT), valeur absolue (ABS), logarithme (LOG), logarithme naturel (LN), exponentiel naturel (EXP), élévation à une puissance (EXPT)

```
(* Initial Settings *)
XMT[1] := 2;
XMT[2] := 7;
N := 2;

(* CRC16 *)
CRCTMP := 16#FFFF;
FOR I := 1 TO N DO
    CRCTMP := CRCTMP XOR XMT[1];
    FOR J := 1 TO 8 DO
        CT := CRCTMP AND 1;
        IF CRCTMP < 0 THEN
            CH := 1;
            CRCTMP := CRCTMP AND 16#7FFF; (* CRCTMP & 0x7FFF *)
        ELSE
            CH := 0;
        END_IF;
        UINT_CRCTMP := WORD_TO_UINT(CRCTMP) / 2;
        CRCTMP := UINT_TO_WORD(UINT_CRCTMP);
        IF CH = 1 THEN
            CRCTMP := CRCTMP OR 16#4000;        (* CRCTMP OR 0x4000 *)
        END_IF;
        IF CT = 1 THEN
            CRCTMP := CRCTMP XOR 16#A001; (* CRCTMP XOR 0xA001 *)
        END_IF;
    END_FOR;
END_FOR;

IF CRCTMP < 0 THEN
    CL := 1;
    CRCTMP := CRCTMP AND 16#7FFF;      (* CRCTMP & 0x7FFF *)

ELSE
    CL := 0;
END_IF;

C_1 := CRCTMP AND 16#FF;                    (* CRCTMP & 0xFF *)
CRCTMP := CRCTMP AND 16#7F00;       (* CRCTMP & 0x7F00 *)
UINT_CRCTMP := WORD_TO_UINT(CRCTMP) / 256;
C_2 := UINT_TO_WORD(UINT_CRCTMP);
```

Référence : la norme IEC 61131-3 est une norme internationale définie par l'IEC (International Electro-technical Commission) qui permet de programmer les automates programmables industriels (API).
La norme est constituée de 7 parties, la partie 3 concernant la programmation des API.

2. Explication du programme cible

L'exemple ci-dessous explique comment créer un programme ST dans un bloc de fonctions afin de calculer la valeur moyenne d'une épaisseur mesurée.



Le type de données doit être défini sur REAL afin de pouvoir stocker les données. Le type REAL accepte les valeurs d'une longueur de 32 bits, dont la plage est la suivante :
-3,402823 x 1038 ~ -1,175494 x 10-38, 0,
+1,175494 x 10-38 ~ +3,402823 x 1038

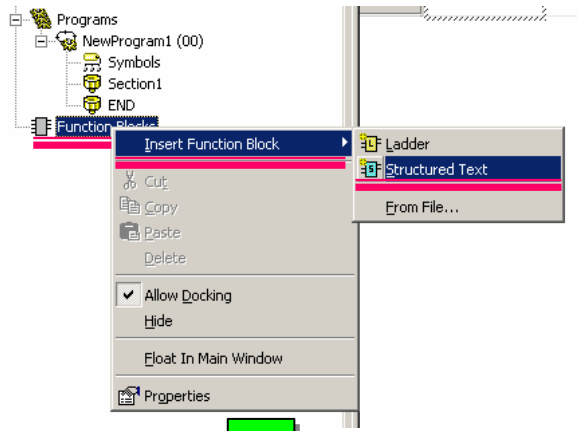
Nom définition FB	AverageCalc_3Value
Symboles d'entrée	x(REAL type), y(REAL type), Z(REAL type)
Symbole de sortie	score(REAL type)
Définition programme ST	score := (x + y + z) / 3.0;

Pour remplacer une valeur par un symbole, utilisez « := ».

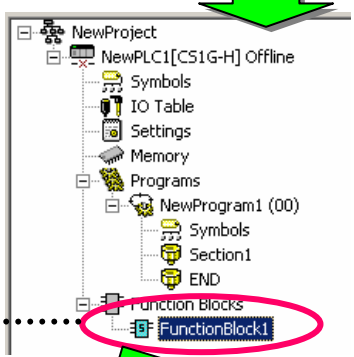
Entrez un point-virgule « ; » pour terminer le code.

3. Création d'un bloc de fonctions à l'aide de ST

Créez un bloc de fonctions à l'aide du langage de texte structuré.

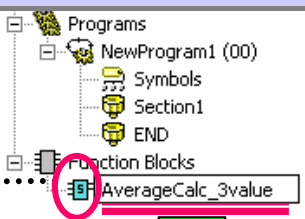


Sélectionnez l'icône de bloc de fonctions à l'aide du curseur, puis cliquez avec le bouton droit.
→ Insérer bloc fonction
→ Texte structuré



Une nouvelle définition de bloc de fonctions est créée.

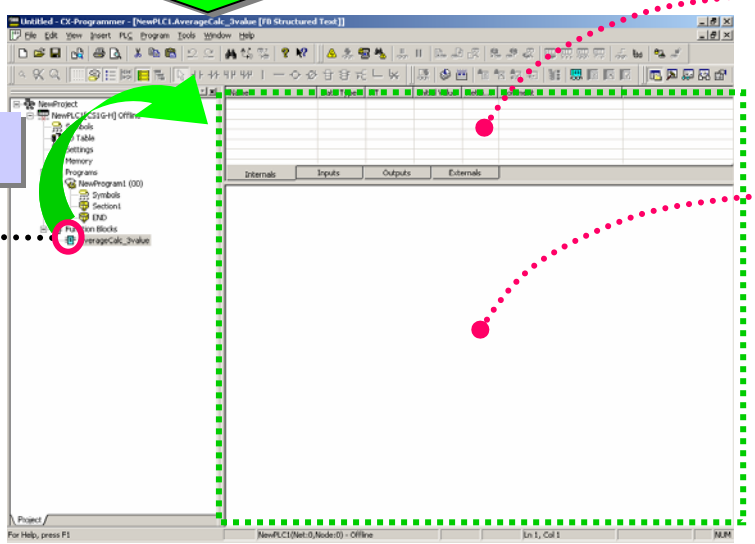
Modifiez le nom de la définition du bloc de fonctions



Sélectionnez l'icône de définition du bloc de fonctions à l'aide du curseur, puis cliquez dessus avec le bouton droit.
Sélectionnez Coller.
→ Renommer
Entrez
[AverageCalc_3value]

Remarque :
Vous ne pouvez pas créer une définition de bloc de fonctions dont le nom commence par un trait de soulignement (« _ »).
N'ajoutez pas « _ » au début du nom.

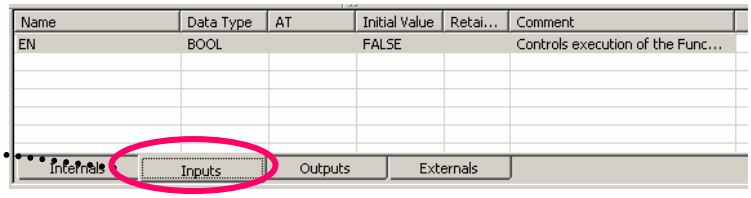
Ouvrez l'Editeur de texte structuré du bloc de fonctions



Sélectionnez l'icône de la définition du bloc de fonctions à l'aide du curseur, puis double-cliquez dessus.

4. Entrée de variables dans des blocs de fonctions

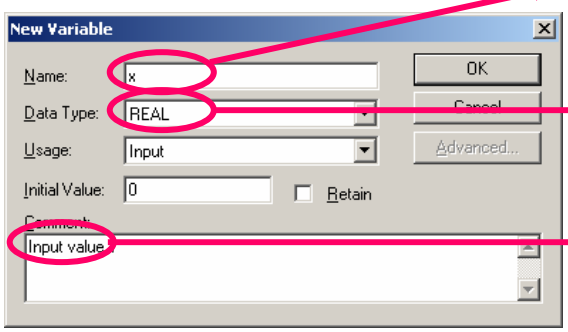
Sélectionnez la table des variables.



Ouvrez l'onglet Entrées.

Sélectionnez Insérer dans le menu contextuel.

Complétez les champs suivants :
Nom
Type de données
Commentaire



Entrez la nom de la variable

Sélectionnez REAL

Entrez un commentaire correspondant

Entrez le symbole d'entrée x, les symboles de sortie y et z en répétant la procédure ci-dessus.

Name	Data Type	AT	Initial Value	Retai...	Comment
EN	BOOL		FALSE		Controls execution of the Func...
x	REAL		0.0		Input value 1
y	REAL		0.0		Input value 2
z	REAL		0.0		Input value 3

Variables d'entrée

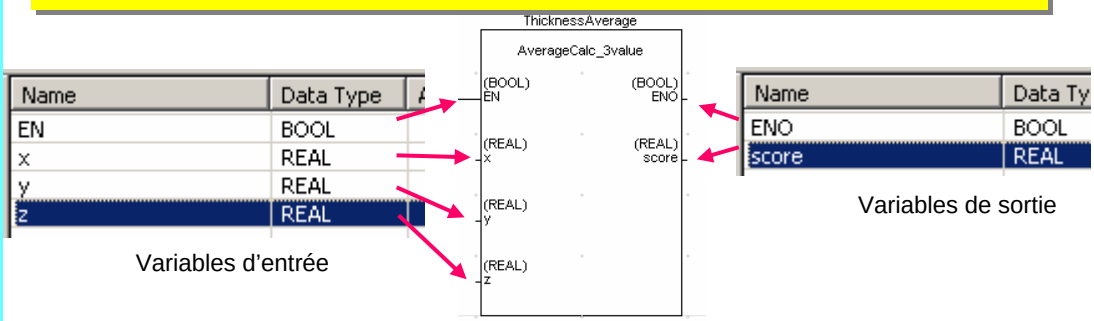
Name	Data Type	AT	Initial Value	Retai...	Comment
ENO	BOOL		FALSE		Indicates successful execution ...
score	REAL		0.0		Avrage

Variables de sortie

Référence : la fonction copier/coller est disponible dans l'en-tête FB.

Référence : l'ordre des variables dans la table FB correspond à l'ordre des paramètres dans l'instance FB (instruction d'appel) dans le schéma de contact normal.

Pour modifier cet ordre, glissez et déposez des variables dans la table.



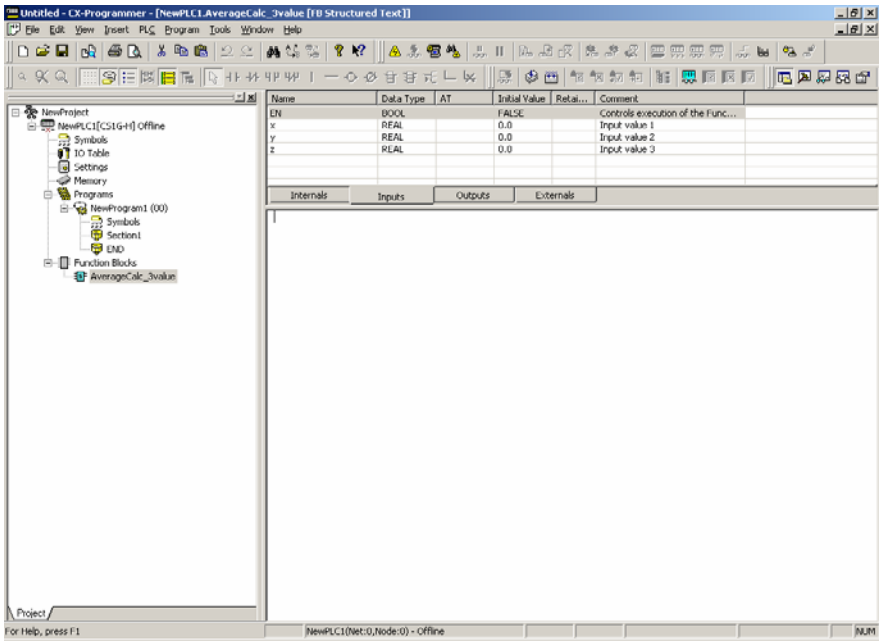
Variables d'entrée

Instance FB (appel)

Variables de sortie

5. Entrée d'un programme ST

Sélectionnez la zone de texte Editeur ST dans la fenêtre Editeur du texte structuré du bloc de fonction.



Entrez le texte suivant dans le champ : « score := (x + y + z) / 3.0; ».

```
score :=(x + y + z) / 3.0;
```

Si l'expression d'entrée est un calcul de type REAL, entrez la valeur constante en utilisant un point comme séparateur et un zéro pour indiquer une décimale (3.0, par exemple).

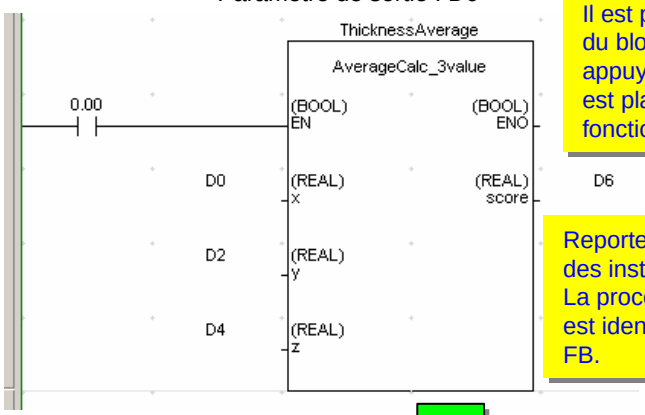
Référence : Il est possible d'entrer des commentaires dans le programme ST. Entrez « (* » et « *) » aux extrémités des chaînes de commentaire (voir ci-dessous). Cela s'avère utile pour enregistrer l'historique des modifications, les expressions de traitement, etc.

```
(* Created by Suzuki 5/21/2004 *)
score := (x + y + z) / 3.0;
```

Remarque: Vous pouvez accéder à une rubrique d'aide affichant la syntaxe de contrôle ST en sélectionnant [Aide ST] dans un menu contextuel de l'Editeur ST.

6. Entrée d'un bloc de fonctions dans le programme et contrôle d'erreurs

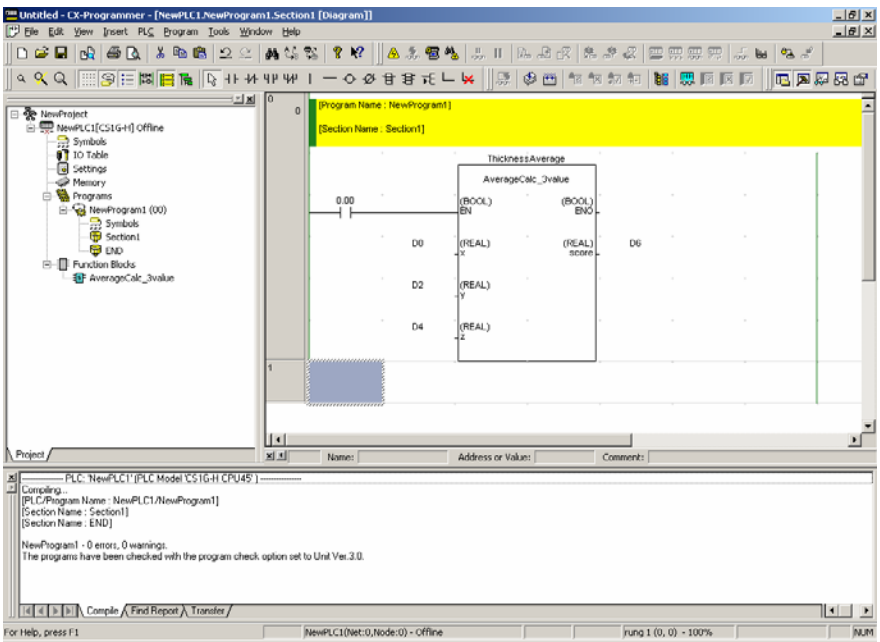
Entrez le FB suivant dans le programme contact.
Nom de l'instance : ThicknessAverage
Paramètres d'entrée : D0, D2, D4
Paramètre de sortie : D6



Il est possible de passer à la définition du bloc de fonctions de référence en appuyant sur [Maj]+[F] lorsque le curseur est placé sur l'instance du bloc de fonctions.

Reportez-vous à la page 2-7 pour entrer des instances FB.
La procédure d'entrée d'instances FB ST est identique à celle d'instances de schéma FB.

Effectuez un contrôle du programme avant de transférer le programme.

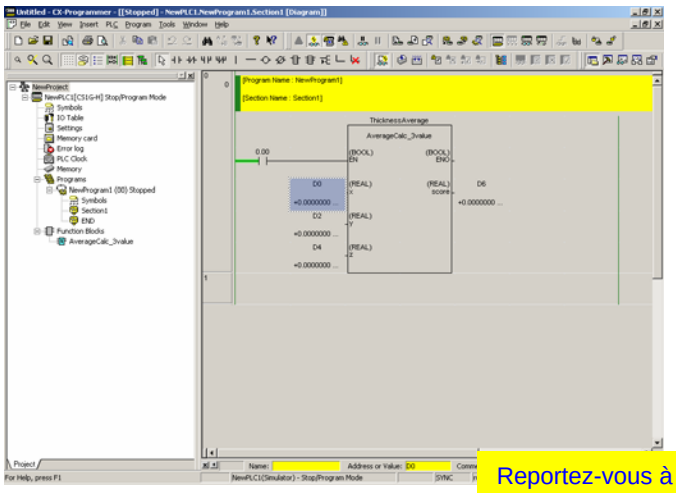


Reportez-vous à la page 2-9 pour le contrôle d'un programme.
La fonction est identique à celle des instances du schéma de blocs de fonctions.

Il est possible de modifier et d'ajouter des variables dans le bloc de fonctions après avoir entré l'instance FB dans l'éditeur de schéma. En cas de modification, l'éditeur de schéma change la couleur de la ligne de terminaison de gauche du segment contenant le bloc de fonctions modifié.
Dans ce cas, sélectionnez l'instance dans l'éditeur de schéma à l'aide du curseur, puis sélectionnez Actualiser invocation de bloc de fonction dans le menu contextuel.

7. Transfert du programme

Activez le mode en ligne de l'API avec CX-Simulator et transférez le programme.

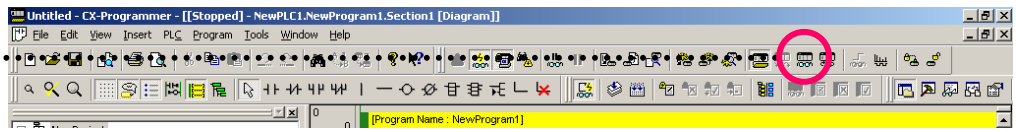


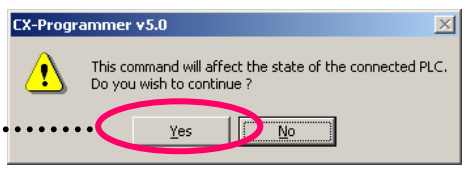
Reportez-vous à la page 2-10 pour savoir comment activer le mode en ligne et transférer le programme.

Faites passer l'API (simulateur) en mode de surveillance.

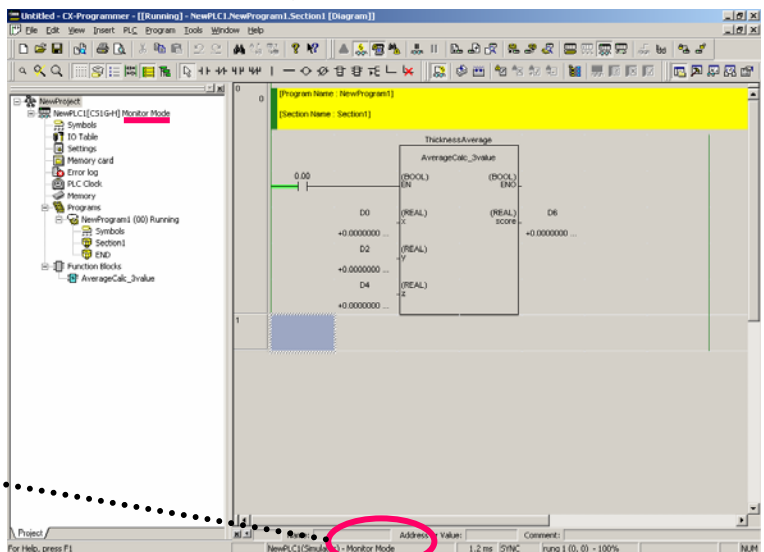
Il est possible de surveiller l'état on/off des contacts et des bobines.

Cliquez sur 





Cliquez sur [Oui]



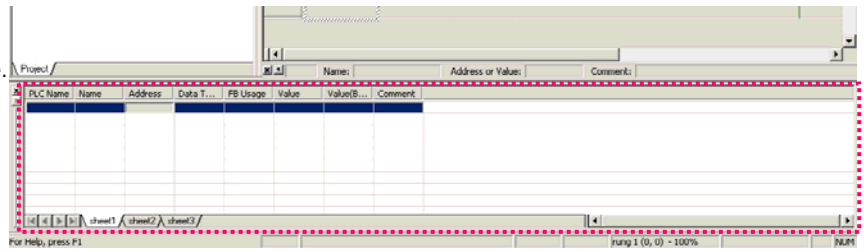
Assurez-vous que l'API est en mode de surveillance.

8. Surveillance de l'exécution d'un bloc de fonctions

Surveillez la valeur actuelle des paramètres dans l'instance FB à l'aide de la fenêtre de surveillance.

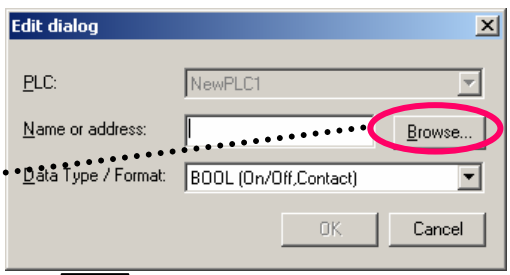
Ouvrez la fenêtre de surveillance.

Alt + 3



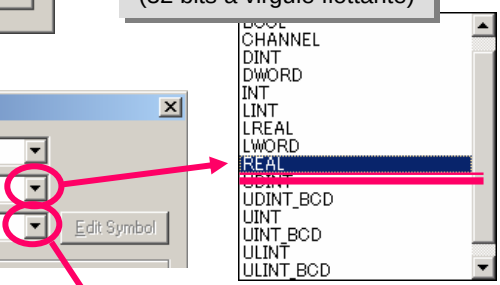
Ouvrez la boîte de dialogue d'édition.

ENT



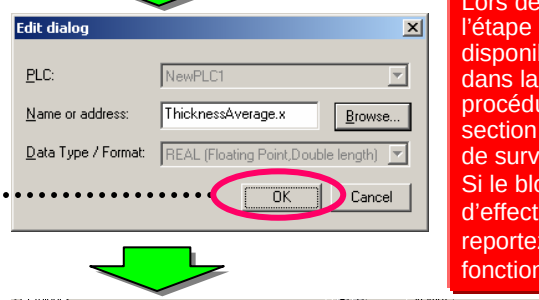
Cliquez sur le bouton Parcourir.

Sélectionnez REAL (32 bits à virgule flottante)



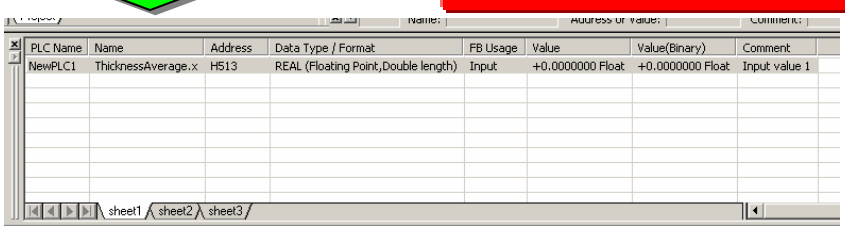
Cliquez sur pour sélectionner une valeur dans les champs suivants :
[Symboles de type]
[Nom ou adresse]

Sélectionnez ThicknessAverage.x



Cliquez sur le bouton [OK].

Lors de la surveillance de variables internes à l'étape de débogage, l'enregistrement collectif est disponible en plus de l'enregistrement individuel dans la fenêtre de surveillance selon la même procédure. Pour plus de détails, reportez-vous à la section 5-8 « Enregistrement par lot dans la fenêtre de surveillance ».
Si le bloc de fonctions est un contact, il est possible d'effectuer une surveillance. Pour plus de détails, reportez-vous à la section 5-5 « Contrôle du fonctionnement - 1 ».



Référence : Exemple d'un programme ST qui utilise IF, THEN, ELSE, END_IF

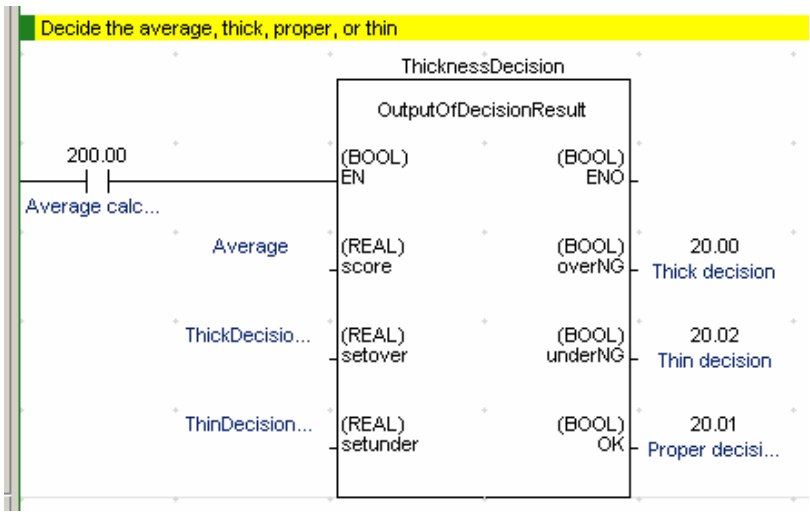
Le programme ST ci-dessous vérifie la valeur moyenne calculée par l'exemple de la page 4-7 par rapport à une plage (limite supérieure ou inférieure).

Définition FB : OutputOfDecisionResult
Symboles d'entrée : score(type REAL), setover(type REAL), setunder(type REAL)
Symboles de sortie : OK(type BOOL), overNG(type BOOL), underNG(type BOOL)

Programme ST :

```
IF score > setover THEN          (* Si score > setover, *)
  underNG := FALSE;              (* Désactiver underNG *)
  OK := FALSE;                   (* Désactiver OK *)
  overNG := TRUE;                (* Activer overNG *)

ELSIF score < setunder THEN      (* Si score =< setover et score < setunder alors *)
  overNG := FALSE;               (* Désactiver overNG *)
  OK := FALSE;                   (* Désactiver OK *)
  underNG := TRUE;               (* Activer underNG *)

ELSE                             (* Si setover > score > setunder alors *)
  underNG := FALSE;              (* Désactiver underNG *)
  overNG := FALSE;               (* Désactiver overNG *)
  OK := TRUE;                    (* Activer OK *)

END_IF;                          (* Fin de la section IF *)
```

Exemple d'une instance FB (nom de l'instance : ThicknessDecision)



Chapitre 5

Fonctions avancées

(création de composants d'un programme à l'aide de FB)

Function Block

1. Vue d'ensemble

Ce chapitre explique comment créer les composants d'un programme utilisateur en fournissant un exemple utilisant des blocs de fonctions.

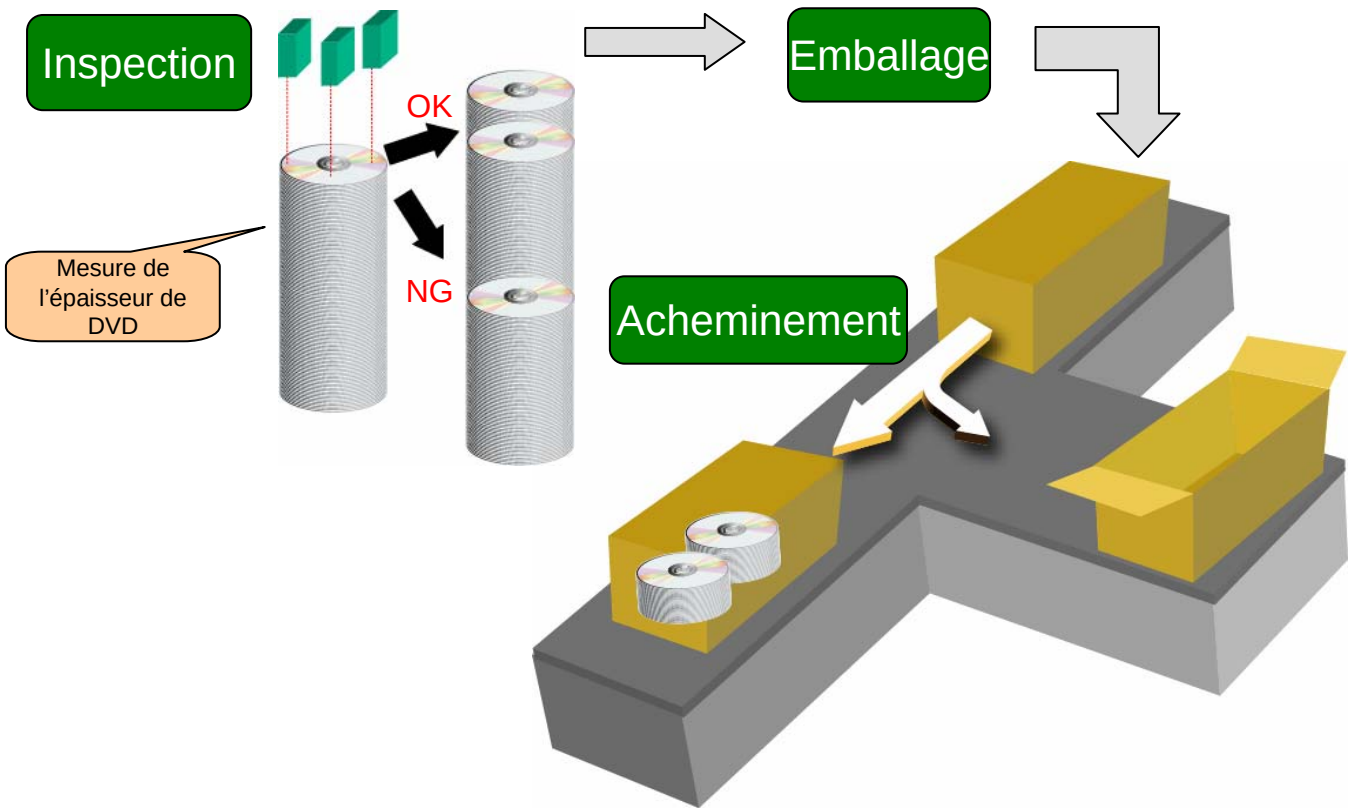
2. Développement d'un programme

Vous trouverez ci-dessous un flux de travail permettant de créer un programme utilisateur à l'aide de composants dans le cadre de l'exemple d'application ci-dessous. Accordez une attention particulière au processus de conception du programme.

(1) Conception du programme
(2) Création des composants
  (2-1) Entrée du composant FB
  (2-2) Débogage du composant FB
  (2-3) Création de la bibliothèque de composants FB (enregistrement de fichier)
(3) Utilisation des composants dans l'application
  (3-1) Importation des composants
  (3-2) Utilisation des composants pour le programme
  (3-3) Débogage du programme
(4) Démarrage

3. Exemple d'application

Voici un exemple d'application concernant une machine d'inspection de DVD.
Le processus peut être divisé en plusieurs étapes : inspection, emballage et acheminement.

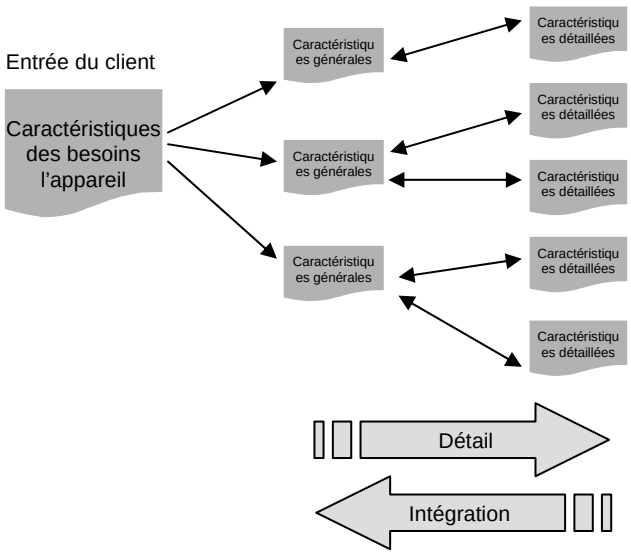


4. Développement d'un programme

L'application peut être matérialisée à l'aide de matériel et d'un logiciel (programme) en combinant des besoins.
Les sections ci-après expliquent comment concevoir le programme à l'aide d'un exemple d'application décrit précédemment.

4-1. Présentation du processus de conception

Les caractéristiques doivent être détaillées et intégrées plusieurs fois afin qu'elles soient réparties et classées conformément au schéma ci-contre.



4-2. Extraction des caractéristiques des besoins

Vous trouverez ci-dessous les caractéristiques des besoins extraites pour cette application.

Vue d'ensemble de la machine d'inspection de DVD (caractéristiques des besoins)
Cond. 1. Le DVD doit être inséré à partir d'un chargeur.
Cond. 2. L'épaisseur du DVD doit être mesurée sur 3 points. L'épaisseur moyenne des mesures doit être calculée. Si elle est comprise dans la plage correspondante, le DVD doit être acheminé vers un stockeur de produits en bon état. Dans le cas contraire, il est acheminé vers un stockeur de produits endommagés.
Cond. 3. Les DVD corrects doivent être emballés dans la caisse.
Cond. 4. Les DVD emballés doivent l'être dans la boîte en carton.
Cond. 5. Les boîtes en carton sont classées en 2 types. Le nombre de commutation doit être compté afin d'évaluer la durée de vie de l'interrupteur fin de course adjacent à l'actionneur de la partie de sélection.
Cond. 6. Autres besoins.
* Pour simplifier la description, ce document se concentre sur une partie de l'appareil (souligné).

4-3. Caractéristiques détaillées et extraction de processus similaires

Le détail des caractéristiques permet de trouver des processus similaires ou universels.

Commande d'actionneur (exemple de processus similaire)
Dans cet exemple, vous pouvez considérer que le contrôle de cylindre pour l'acheminement des produits corrects et endommagés et le contrôle d'actionneur pour l'acheminement des boîtes en carton sont identiques. Vous trouverez ci-dessous des besoins extraits pour ces processus.

- Le processus est constitué de 2 actionneurs qui effectuent un mouvement bilatéral en entrée.
- Le fonctionnement de chaque sens doit être verrouillé.
- Le processus dispose d'un signal d'entrée permettant de réinitialiser le fonctionnement.

Contrôle de seuil moyen (exemple de processus universel)
Un processus doit être extrait pour une utilisation universelle, même s'il n'est utilisé qu'une seule fois pour l'application. Dans cet exemple, un processus est extrait afin de calculer la moyenne de 3 épaisseurs mesurées de DVD et de contrôler si elle est comprise dans le seuil. Vous trouverez ci-dessous des besoins extraits pour ce processus.

- La moyenne des 3 mesures doit être calculée.
- La valeur moyenne doit être contrôlée afin de déterminer si elle est comprise entre les limites inférieure et supérieure du seuil.

Ces besoins constituent la base des composants. Les noms des composants sont définis en tant que FB « ActuatorControl » et « AvgValue_ThresholdCheck ».

4-3-1. Création des caractéristiques des composants

La réutilisation de composants permet d'améliorer la productivité du développement de programme. Pour simplifier la réutilisation, il est important de créer des caractéristiques et d'insérer des commentaires facilitant la compréhension des caractéristiques d'entrée/sortie ou d'utilisation sans devoir consulter le composant.
Il est recommandé de décrire une référence pour la bibliothèque FB OMRON.

4-3-2. Exemple de création d'un composant FB

FB « ActuatorControl »

Il doit être décrit dans une séquence de schéma de contact car il s'agit d'un processus de contrôle séquentiel.

[Variables d'entrée]

Name	Data Type	AT	Initial Value	Retained	Comment
EN	BOOL		FALSE		Controls execution of the Function Block.
PosDirInput	BOOL		FALSE		Input for positive direction
NegDirInput	BOOL		FALSE		Input for negative direction
LSpos	BOOL		FALSE		Limit switch for positive direction
LSneg	BOOL		FALSE		Limit switch for negative direction

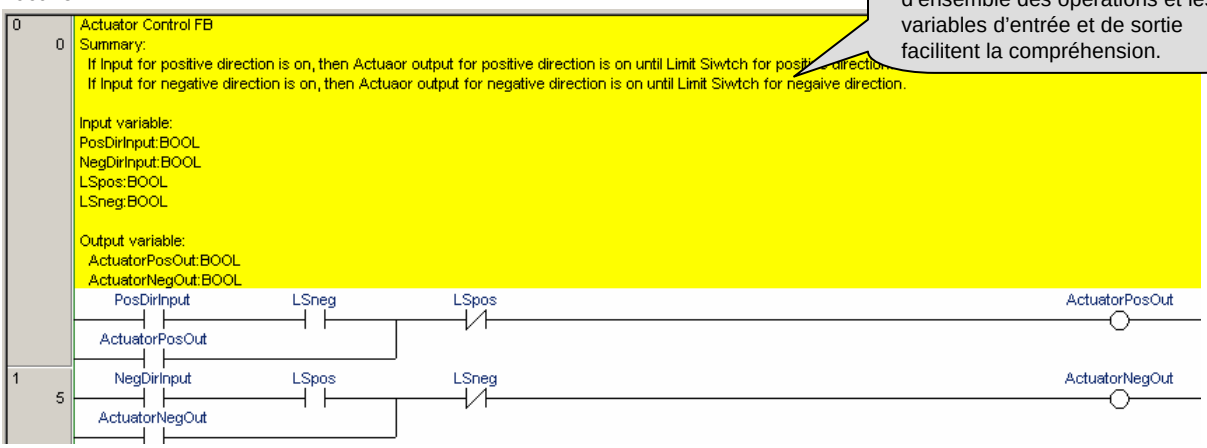
[Variables de sortie]

Name	Data Type	AT	Initial Value	Retained	Comment
ENO	BOOL		FALSE		Indicates successful execution of the Function Block ...
ActuatorPosOut	BOOL		FALSE		Actuator output for positive direction
ActuatorNegOut	BOOL		FALSE		Actuator output for negative direction

[Variables internes]

Aucune.

Les commentaires pour la vue d'ensemble des opérations et les variables d'entrée et de sortie facilitent la compréhension.



FB « AvgValue_ThresholdCheck »

Il doit être décrit à l'aide du langage ST car il s'agit d'un processus de calcul numérique et de comparaison.

[Variables d'entrée]

Name	Data Type	AT	Initial Value	Retained	Comment
EN	BOOL		FALSE		Controls execution of the Function Block.
Input1	REAL		0.0		Input value 1
Input2	REAL		0.0		Input value 2
Input3	REAL		0.0		Input value 3
UpLimit	REAL		0.0		Upper limit value
LowLimit	REAL		0.0		Lower limit value

[Variables de sortie]

Name	Data Type	AT	Initial Value	Retained	Comment
ENO	BOOL		FALSE		Indicates successful execution of the Function Block...
Result	BOOL		FALSE		OK or NG judge flag

[Variables internes]

Name	Data Type	AT	Initial Value	Retained	Comment
AvgValue	REAL		0.0		

```
(* Agarage value calculation and check of threshould for three values *)

AvgValue := ( Input1 + Input2 + Input3 ) / 3.0;              (* Divides Input 3 values by 3 *)
IF ((AvgValue <=UpLimit) AND (AvgValue >=LowLimit)) THEN (* Compare the agarage value if below of upper limit or above of lower limit *)
    Result := TRUE;
    ELSE
    Result := FALSE;
END_IF;
```

Remarque : définissez des noms génériques de longueur suffisante pour les FB et les variables du schéma de contact et du texte structuré, au lieu de noms spécifiques pour les fonctions lors de la création.

4-4. Intégration des FB

Les composants détaillés du processus sont à présent extraits. Vous allez créer les composants de l'application en les combinant dans les sections suivantes.

4-4-1. Combinaison de composants - DVD_ThickSelectControl

Le besoin 2 « L'épaisseur du DVD doit être mesurée sur 3 points. L'épaisseur moyenne des mesures doit être calculée. Si elle est comprise dans la plage correspondante, le DVD doit être acheminé vers un stockeur de produits en bon état. Dans le cas contraire, il est acheminé vers un stockeur de produits endommagés. » peut être considérée comme un processus qui associe les FB « AvgValue_ThresholdCheck » et « ActuatorControl » présentés dans la section précédente. La « combinaison » de ces composants permet de créer le FB de composant intégré « DVD_ThickSelectControl ». Voici un exemple de FB à créer.

[Variables d'entrée]

Name	Data Type	AT	Initial Value	Retained	Comment
EN	BOOL		FALSE		Controls execution of the Function Block.
LSright	BOOL		FALSE		Limit switch for cylinder right direction
LSleft	BOOL		FALSE		Limit switch for cylinder left direction
Measure1	REAL		0.0		Measurement result 1 of DVD thickness (mm)
Measure2	REAL		0.0		Measurement result 2 of DVD thickness (mm)
Measure3	REAL		0.0		Measurement result 3 of DVD thickness (mm)

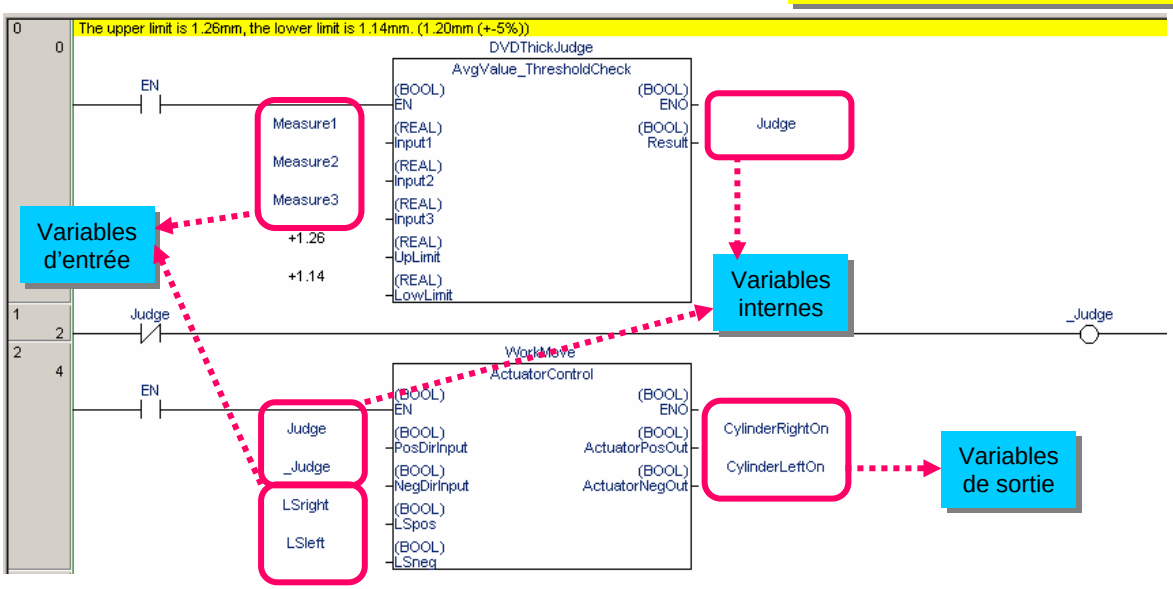
[Variables de sortie]

Name	Data Type	AT	Initial Value	Retained	Comment
ENO	BOOL		FALSE		Indicates successful execution of the Function Block ...
CylinderRightOn	BOOL		FALSE		Output for sylinder right direction
CylinderLeftOn	BOOL		FALSE		Output for sylinder left direction

[Variables internes]

Name	Data Type	AT	Initial Value	Retained	Cor
WorkMove	FB [ActuatorControl]				
DVDThickJudge	FB [AvgValue_ThresholdCheck]				
Judge	BOOL		FALSE		
_Judge	BOOL		FALSE		

Ce FB possède un nom spécifique, ainsi que des noms de variables qui contiennent les mots « DVD » ou « Cylinder », car il est créé particulièrement pour une application.



Un bloc de fonctions peut être appelé à partir d'un autre bloc de fonctions. Ce principe est appelé « imbrication ».

Pour effectuer une imbrication, déclarez une variable de type FUNCTION BLOCK (FB) comme variable interne afin d'utiliser le nom de la variable comme instance.

4-4-2. Ajout de fonctions aux composants - WorkMoveControl_LSONcount

La condition 5 « Les boîtes en carton sont classées en 2 types. Le nombre de commutation doit être compté afin d'évaluer la durée de vie de l'interrupteur fin de course adjacent à l'actionneur de la partie de sélection. » peut être matérialisée en déterminant le nombre de commutations OFF ® ON d'un interrupteur fin de course en tant qu'entrée pour « ActuatorControl ». Ce composant est appelé FB « WorkMoveControl_LSONcount ». Voici un exemple de FB à créer.

[Variables d'entrée]

Name	Data Type	AT	Initial Value	Retained	Comment
EN	BOOL		FALSE		Controls execution of the Function Block.
RightDirInput	BOOL		FALSE		Condition to move actuator to right direction
LeftDirInput	BOOL		FALSE		Condition to move actuator to left direction
LSright	BOOL		FALSE		Limit switch for acutuator right direction
LSleft	BOOL		FALSE		Limit switch for acutuator left direction
Reset	BOOL		FALSE		Resets number of times for opening - closing li...

[Variables de sortie]

Name	Data Type	AT	Initial Value	Retained	Comment
ENO	BOOL		FALSE		Indicates successful execution of the Functio...
ActuatorRightOn	BOOL		FALSE		Output for actuator right direction
ActuatorLeftOn	BOOL		FALSE		Output for actuator left direction
LS_ONnumber	LINT		0		

[Variables internes]

Name	Data Type	AT	Initial Value	Retained	Comment
PrevCycleLS	BOOL		FALSE		
WorkMove	FB [ActuatorControl]				

```
(* Work move control and count of number of times open - close of limit swvitch *)
(* Created by: machine development div. Yamada: 10-01-2005 *)

(* Resets number of times opening - closing limit siwtch *)
IF Reset = TRUE THEN
      PrevCycleLS := FALSE;
END_IF;


(* Calls WorkMove (instance of ActuatorControl FB) *)
WorkMove(RightDirInput, LeftDirInput, LSright, LSleft, ActuatorRightOn, ActuatorLeftOn);

(* Counts number of times opening - closing limit switch *)
IF PrevCycleLS = FALSE and LSright = TRUE THEN
      LS_ONnumber := LS_ONnumber+1;
END_IF;
PrevCycleLS := LSright;  (* Copies LSright to compare at next execution *)
```

FB schéma appelé dans ST.

Comment appeler un bloc de fonctions (FB) en langage structuré

FB à appeler : MyFB
Variable E/S du FB à appeler :
  Entrée : Input1, Input2
  Sortie : Output1, Output2

Instance de MyFB déclarée dans ST : MyInstance
Variable E/S à transmettre au FB dans ST :
  Entrée : STInput1, STInput2
  Sortie : STOutput1, STOutput2

Dans cet exemple, l'appel de l'instance FB depuis ST doit être décrit comme suit :
  MyInstance(Input1 := STInput1, Input2 := STInput2, Output1 => STOutput1, Output2 => STOutput2);

Une fois toutes les variables d'entrée/sortie décrites, vous pouvez omettre la description des variables et des opérateurs d'affectation à appeler.
  MyInstance(STInput1, STInput2, STOutput1, STOutput2);

En décrivant les variables et les opérateurs d'affectation à appeler, vous ne pouvez décrire qu'une partie des variables d'entrée/sortie.
  MyInstance(Input1 := STInput1, Output2 => STOutput2);

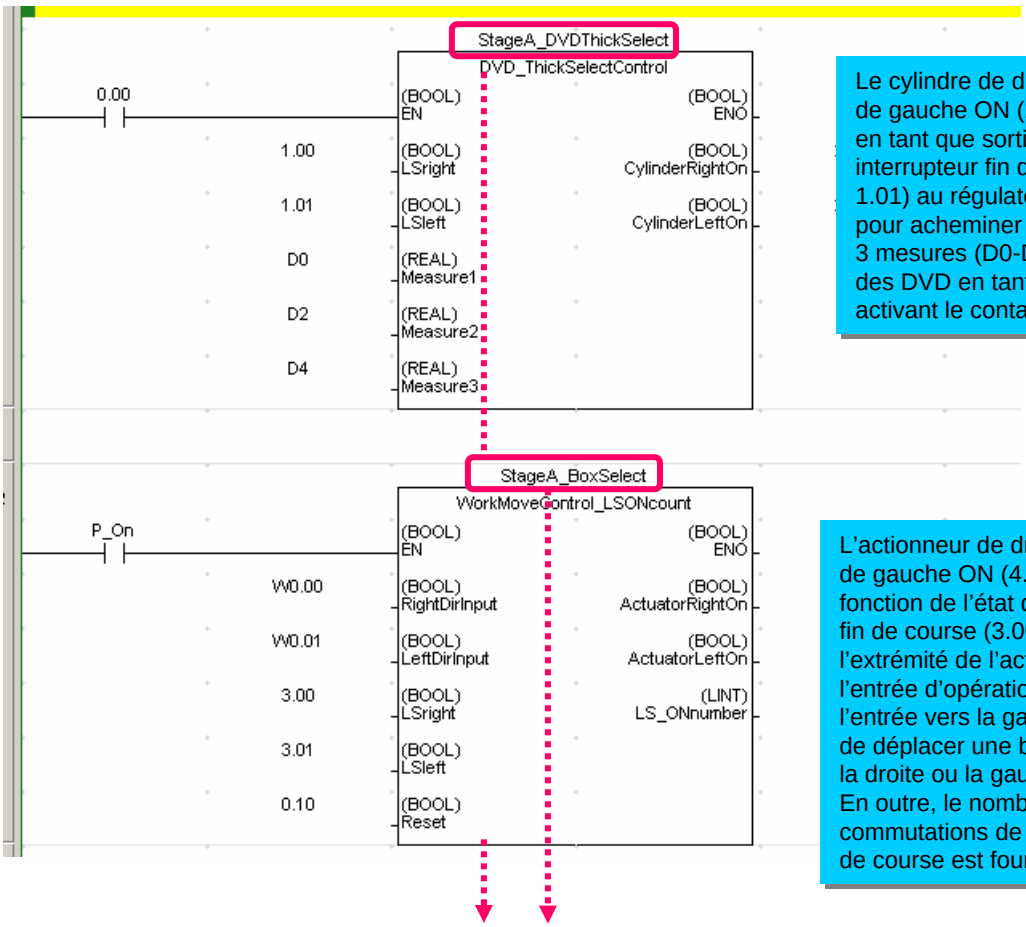
4-5. Description complète du programme

Pour que les composants (FB) présentés ici puissent fonctionner en tant que programme, un circuit doit être créé afin d'appeler un composant intégré depuis le programme principal.
* L'exemple ci-dessous est limité aux conditions 2 et 5.

[Variables globales]

Name	Data Type	Address / Value	Rack Location	Usage
StageA_BoxSelect	FB [WorkMoveControl_LSONcount]	N/A [Auto]		
StageA_DVDThickSelect	FB [DVD_ThickSelectControl]	N/A [Auto]		

* Les variables d'instance autres que celles utilisées pour FB sont omises.



Le cylindre de droite ON (2.00) ou de gauche ON (2.01) fonctionne en tant que sortie à l'aide d'un interrupteur fin de course (1.00, 1.01) au régulateur de cylindre pour acheminer les DVD et les 3 mesures (D0-D5) de l'épaisseur des DVD en tant qu'entrées et en activant le contact 0.00.

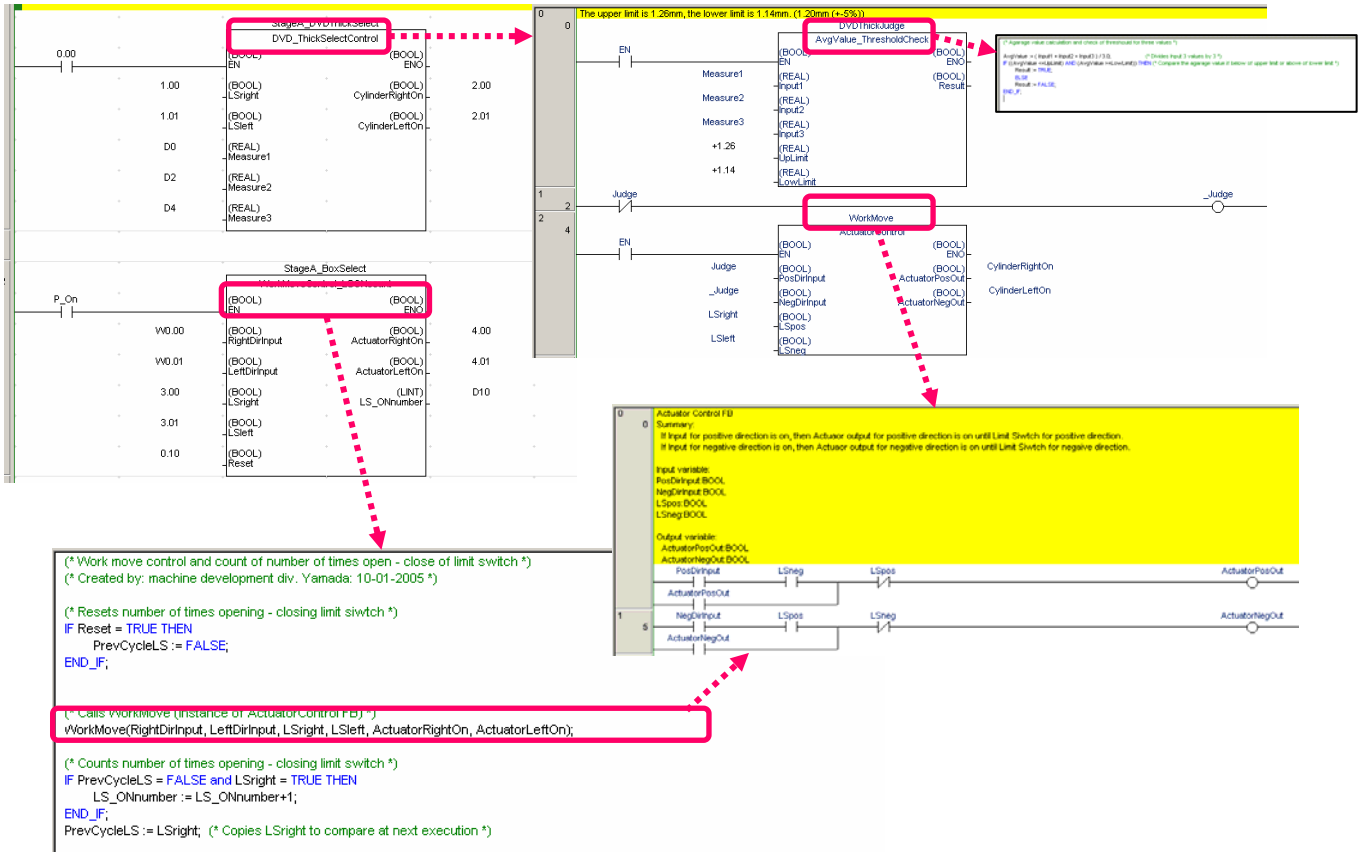
L'actionneur de droite ON (4.00) ou de gauche ON (4.01) s'active en fonction de l'état d'un interrupteur fin de course (3.00, 3.01) à l'extrémité de l'actionneur lors de l'entrée d'opération (W0.00) et l'entrée vers la gauche (W0.01) afin de déplacer une boîte de DVD vers la droite ou la gauche.
En outre, le nombre de commutations de l'interrupteur fin de course est fourni à D10-13.

Pourquoi l'instance porte le nom « StageA* » ?**
Même si cela n'est pas explicitement expliqué dans l'exemple d'application, un programme correspondant à la nouvelle étape B peut être créé uniquement en décrivant une instance « StageB*** » dans le programme et en définissant les paramètres nécessaires sans devoir enregistrer un nouveau bloc de fonctions.
Un bloc de fonctions peut posséder plusieurs instances. A l'aide d'une définition de FB vérifiée par opération (algorithme), il est possible de créer un programme uniquement en lui attribuant une adresse.

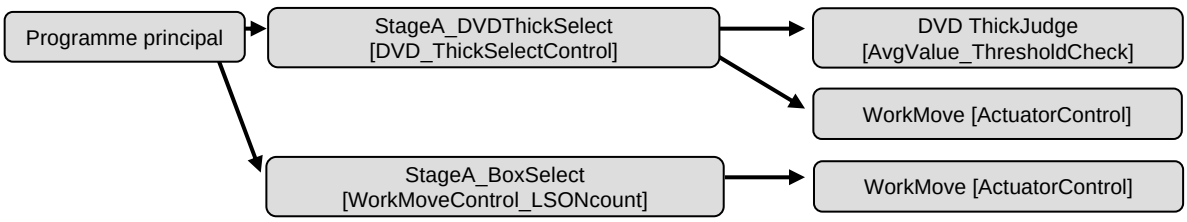
4-5-1. Structure complète du programme

Cette section vérifie la structure du programme, dont les composants (blocs de fonctions).

[Programme principal]

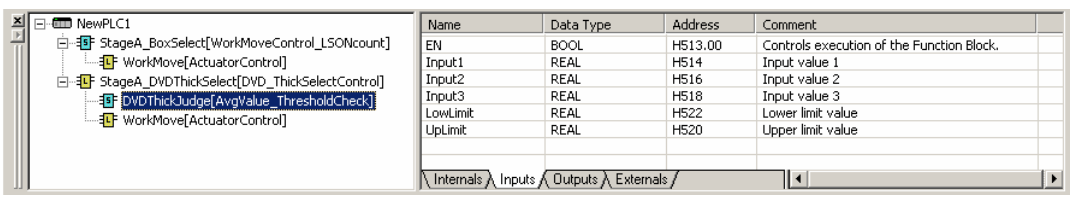


Les noms des instances et des FB peuvent se présenter comme suit : (Le nom de chaque FB est placé entre [ ].)



Dans un programme structuré, plus particulièrement visant à modifier un composant de niveau inférieur (FB), il est important de comprendre la relation parent/enfant et le partage des composants si le flux de traitement doit être effacé en cas de débogage, etc. Il est recommandé de créer un schéma clair et précis de la structure complète du programme dans le cadre de la conception.

CX-Programmer version 6.0 affiche le visualiseur d'instance FB lorsque vous appuyez sur [Alt]+[5]. Vous pouvez ainsi comprendre facilement la structure du programme représentée par les FB. En outre, il est possible de vérifier l'adresse attribuée aux instances FB.



5. Entrée d'une définition FB

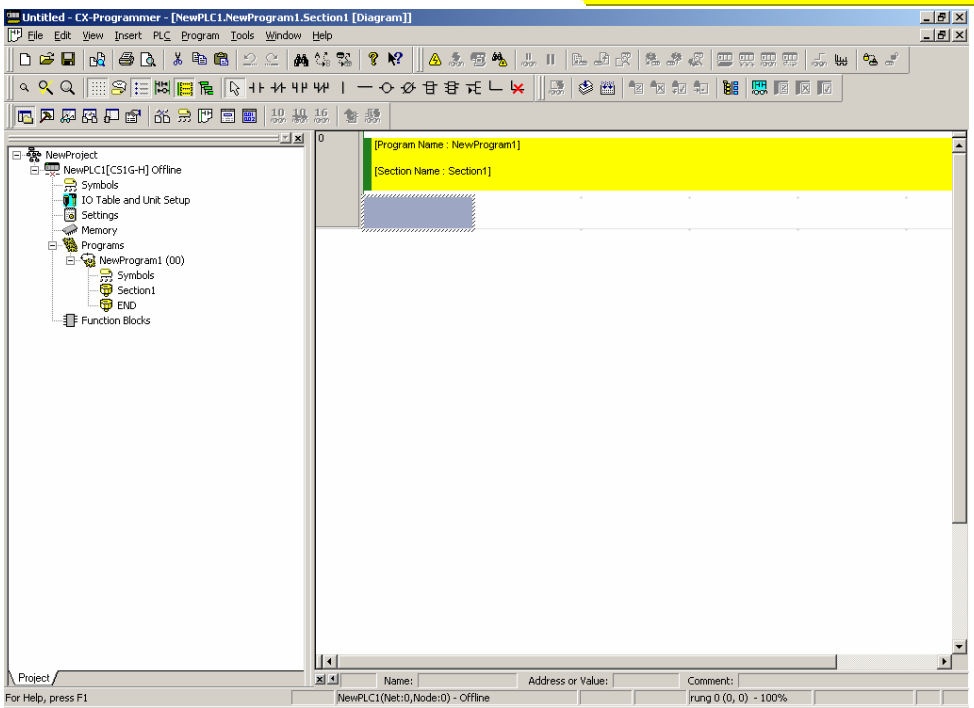
Cette section explique comment entrer et déboguer un programme.

Un nouveau projet doit être créé et le FB « ActuatorControl » de la page 5-4 doit être entré.

5-1. Création d'un projet et configuration du modèle d'API et du type d'UC

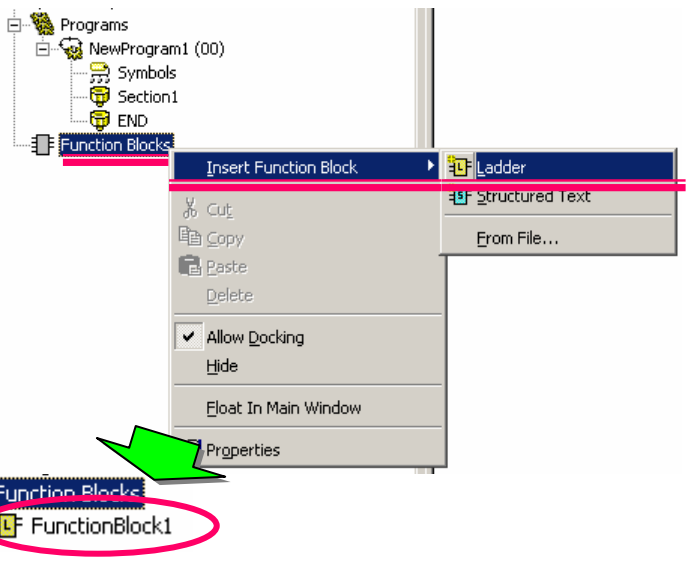
Reportez-vous à la page 2-3 pour créer un projet.

! Sélectionnez un modèle d'API pour utiliser les blocs de fonctions : CS1G-H, CS1H-H, CJ1G-H, CJ1H-H, CJ1M



5-2. Création d'un FB de définition de schéma

Créez un FB de définition de schéma.

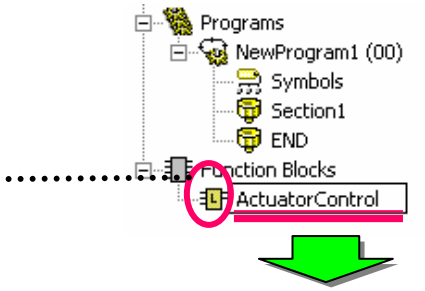


Placez le curseur sur une icône de bloc de fonctions , puis cliquez dessus avec le bouton droit. Sélectionnez
→ Insérer bloc fonction
→ Contact

Le FB est à présent créé.

5-3. Entrée du programme FB

Modifiez le nom d'une définition FB.

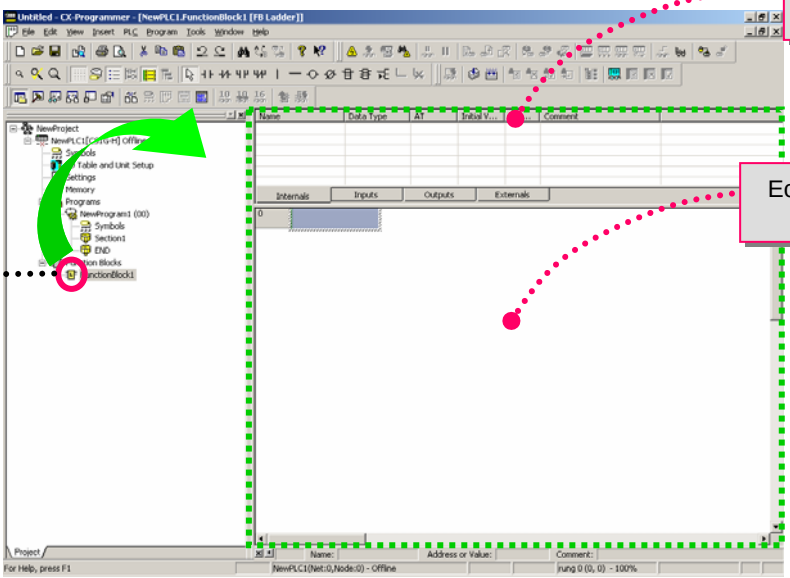


Placez le curseur sur une icône de bloc de fonctions copié, puis cliquez dessus avec le bouton droit. Sélectionnez → Renommer Entrez [ActuatorControl].

Attention :
Vous ne pouvez pas créer une définition de bloc de fonctions dont le nom commence par « _ ».
Le nom doit commencer par un caractère autre que « _ ».

Ouvrez l'Editeur contact FB.

Placez le curseur sur une icône de bloc de fonctions, puis double-cliquez dessus pour ouvrir l'Editeur de texte structuré du bloc de fonction.



Sélectionnez la table des variables et enregistrez les variables dans le bloc de fonctions.
Toutes les variables du FB « ActuatorControl » de la page 5-4 doivent être enregistrées.

Remarque : L'ordre des variables doit être identique à celui des instances FB.
Pour modifier cet ordre, sélectionnez le nom d'une variable, puis effectuez un glisser/déplacer.

Sélectionnez l'écran d'entrée de contact, puis entrez un programme de contact.
Toutes les variables du FB « ActuatorControl » de la page 5-4 doivent être enregistrées.

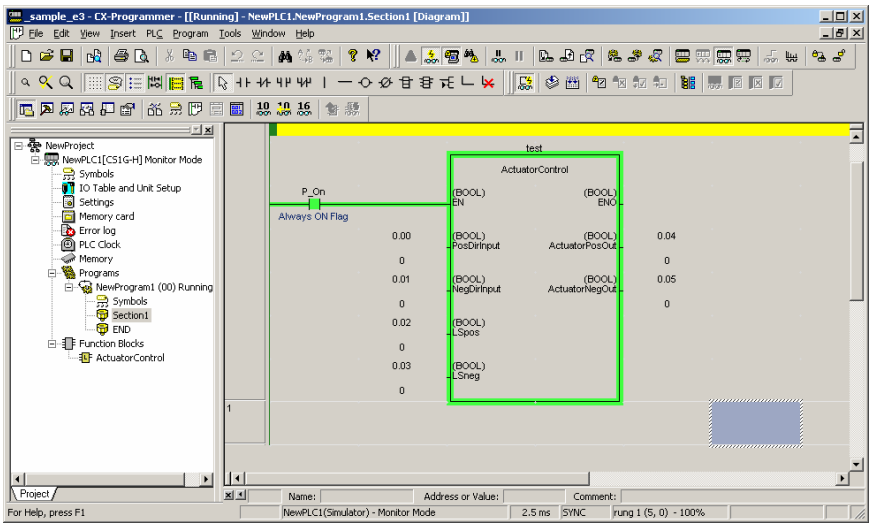
Remarque : Vous pouvez entrer un circuit dans l'Editeur contact FB de la même manière que dans l'éditeur contact principal. Toutefois, l'entrée d'une adresse dans le FB n'est pas valide.

Remarque : Pour entrer une liste de variables dans un commentaire, sélectionnez une variable de la table, puis copiez-la. Vous pouvez l'utiliser pour optimiser l'entrée.

5-4. Transfert du programme

Connectez-vous à CX-Simulator en ligne, transférez un programme, puis activez le mode de surveillance le l'API (simulateur).

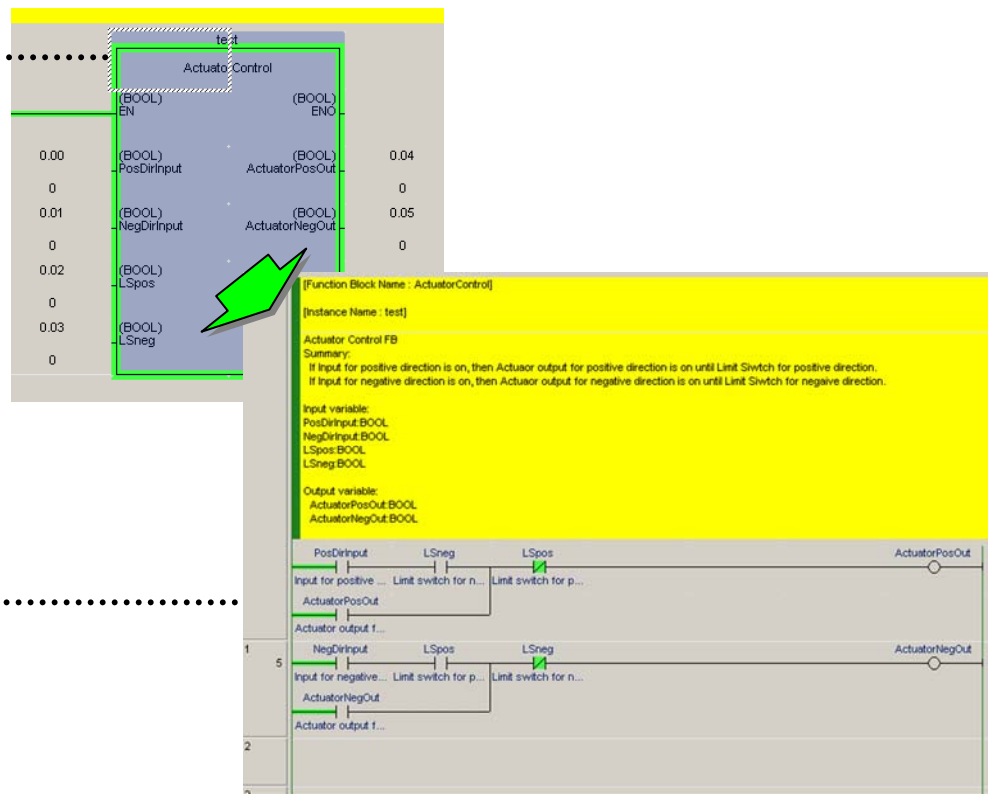
Pour savoir comment se connecter en ligne et transférer un programme, reportez-vous à la page 2-10.



5-5. Vérification du fonctionnement - 1

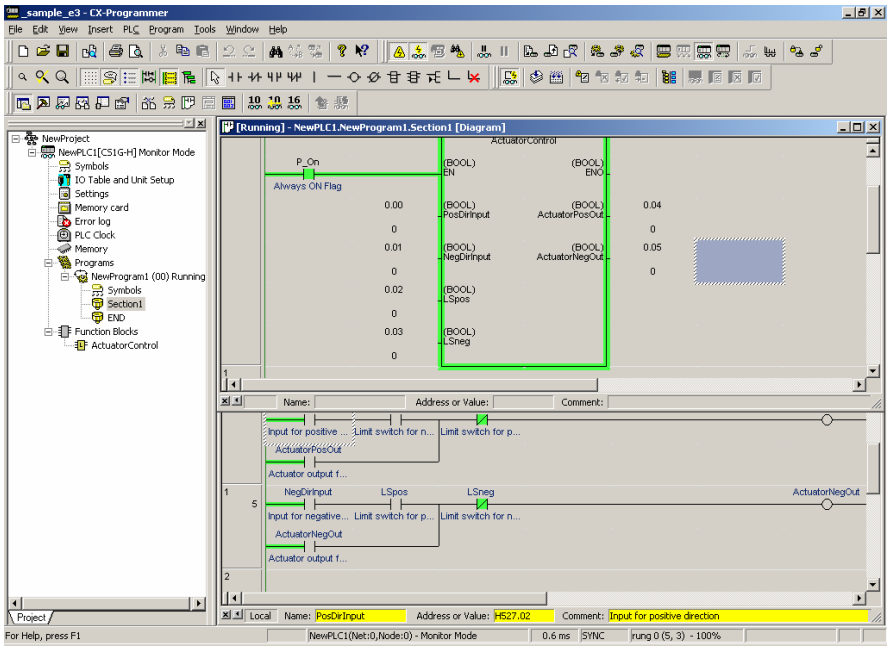
Modifiez la valeur actuelle du paramètre de l'instruction d'appel FB dans le contact principal, puis vérifiez le fonctionnement du FB « ActuatorControl ».
Surveillez d'abord l'instance du FB ActuatorControl.

Placez le curseur sur l'instruction d'appel FB, puis double-cliquez dessus ou cliquez sur le bouton  .



L'instance de contact FB (l'adresse doit être attribuée) est surveillée.

Affichez simultanément le contact principal et l'instance FB (contact FB appelé par le contact principal), puis vérifiez le fonctionnement lors de la modification de la valeur actuelle du paramètre de l'instruction d'appel FB dans le contact principal.

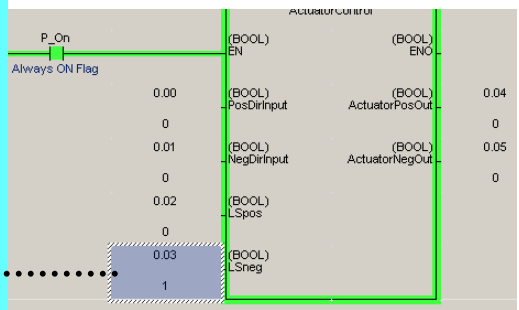


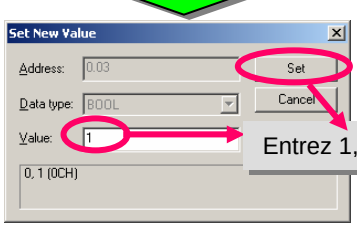
5-6. Vérification du fonctionnement - 2

Entrez les valeurs de paramètre suivantes de l'instruction d'appel FB, puis vérifiez si la sortie prévue est fournie. Dans cet exemple, seule la valeur (1) s'affiche, mais toutes les combinaisons de conditions doivent être vérifiées.

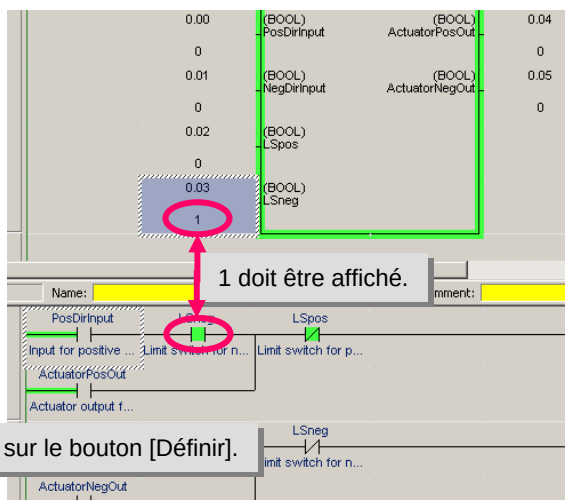
(1) Etat initial : activez 0.03. => 0.04 et 0.05 doivent être désactivés. L'écran de surveillance de contact d'instance FB doit présenter un état correspondant à la valeur.
(2) Fonctionnement du sens avant de l'actionneur - 1 : activez 0.00 => 0.04 doit être activé. L'écran de surveillance de contact d'instance FB doit présenter un état correspondant à la valeur.
(3) Fonctionnement du sens avant de l'actionneur - 2 : désactivez 0.00 => 0.04 doit être activé et 0.05 désactivé. L'écran de surveillance de contact d'instance FB doit présenter un état correspondant à la valeur.
(4) Fonctionnement du sens avant de l'actionneur - 3 : activez 0.02 => 0.04 et 0.05 doivent être désactivés. L'écran de surveillance de contact d'instance FB doit présenter un état correspondant à la valeur.

Placez le curseur sur 0.03, puis appuyez sur la touche [ENT].





Entrez 1, puis cliquez sur le bouton [Définir].



5-7. Entrée/Débogage des autres définitions FB

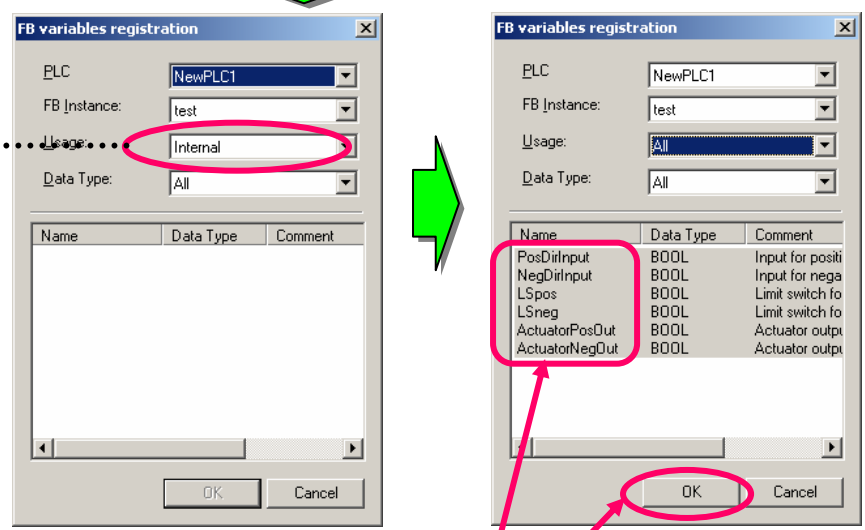
Vous avez précédemment appris à entrer et déboguer le FB « ActuatorControl », mais vous devez également entrer et déboguer les autres définitions FB.

5-8. Enregistrement par lot dans la fenêtre de surveillance

A des fins de débogage, vous pouvez utiliser l'enregistrement par lot de l'adresse d'instance FB dans la fenêtre de surveillance au lieu de la surveillance de schéma FB.

Placez le curseur sur l'instruction d'appel FB à enregistrer, cliquez dessus avec le bouton droit, puis sélectionnez [Enregistrer dans une fenêtre Visu. dynamique] dans le menu contextuel qui s'affiche.

Si nécessaire, sélectionnez l'utilisation et le type de données.



Sélectionnez le nom à enregistrer, puis cliquez sur le bouton [OK].

5-9. Exécution pas à pas à l'aide de la fonction de simulation

Grâce à la définition d'un point d'interruption de la fonction de simulation et à la fonction d'exécution pas à pas, vous pouvez interrompre l'exécution du programme et consulter facilement l'état de traitement lors de l'exécution du programme.

Cette fonction peut être utilisée avec CX-One version 1.1 ou ultérieure (CX-Programmer version 6.1, CX-Simulator version 1.6 ou ultérieure).

5-9-1. Explication des boutons de simulation

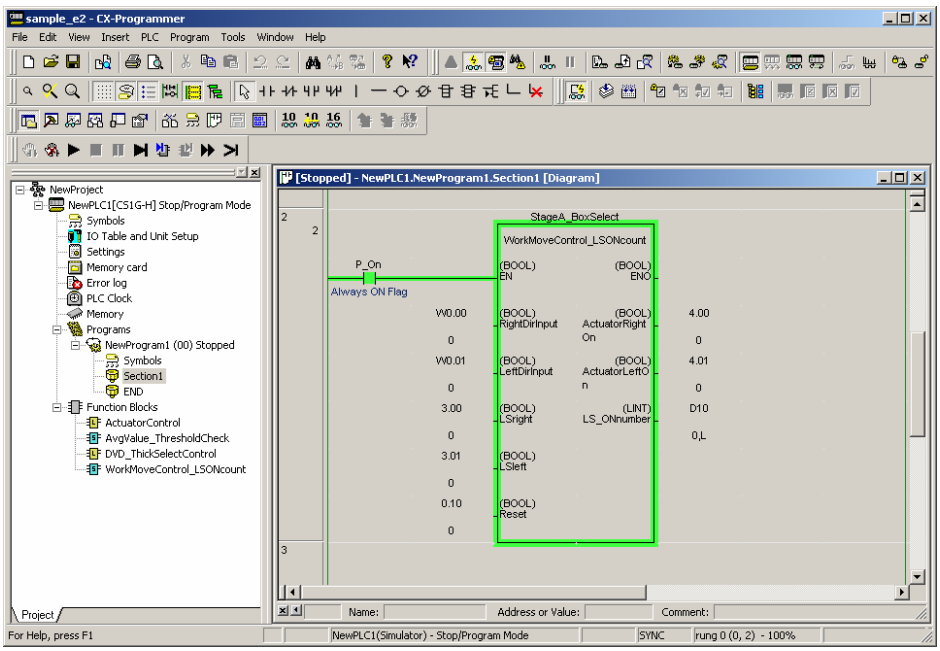
Les boutons de barre d'outils ci-dessous sont utilisés pour la fonction de simulation. Vous trouverez ci-dessous l'explication de chacun de ces boutons.

Boutons de simulation

	Définir/effacer le point d'arrêt (F9)	Sélectionne les emplacements (schéma, ST) où vous voulez insérer un point d'arrêt lors de l'exécution de la simulation. Un repère rouge s'affiche lorsque vous cliquez sur ce bouton.
	Effacer tous les points d'interruption	Supprime un point d'arrêt (repère rouge) défini à l'aide du bouton Définir le point d'arrêt.
	Run (mode surveillance) (F8)	Exécute le programme utilisateur. Le mode d'exécution se change en mode de surveillance.
	Stop (mode programme)	Arrête l'exécution du programme utilisateur. Le mode d'exécution se change en mode de programme.
	Pause	Met en pause l'exécution du programme utilisateur au niveau du curseur.
	Exécution pas (F10)	Exécute un pas du programme utilisateur. Dans le cas d'un schéma, une instruction, et dans le cas de texte structuré, une ligne.
	Pas In (F11)	Exécute un pas du programme utilisateur. Si l'emplacement du curseur appelle l'instruction d'appel FB, il y a transfert vers l'instance FB appelée (schéma ou ST).
	Pas Out (Maj+F11)	Exécute un pas du programme utilisateur. Si l'emplacement du curseur correspond à l'instance FB, il y a transfert vers l'instruction d'appel FB.
	Exécution par pas continus	Exécute les pas du programme utilisateur de manière continue en marquant une pause déterminée.
	Exécution d'un balayage	Exécute un balayage du programme utilisateur (un cycle).

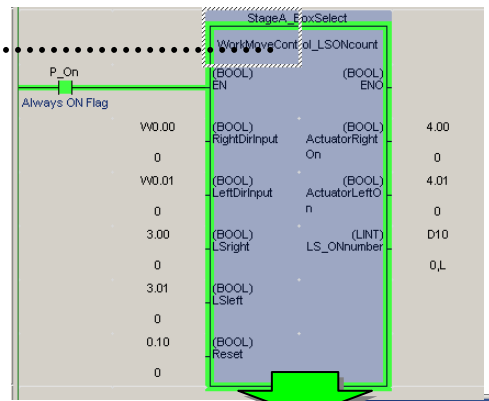
5-9-2. Définition du point d'interruption et exécution des pas

Vous trouverez ci-dessous une explication basée sur un exemple de fonction de simulation à l'aide du FB « WorkMoveControl_LSONcount ».

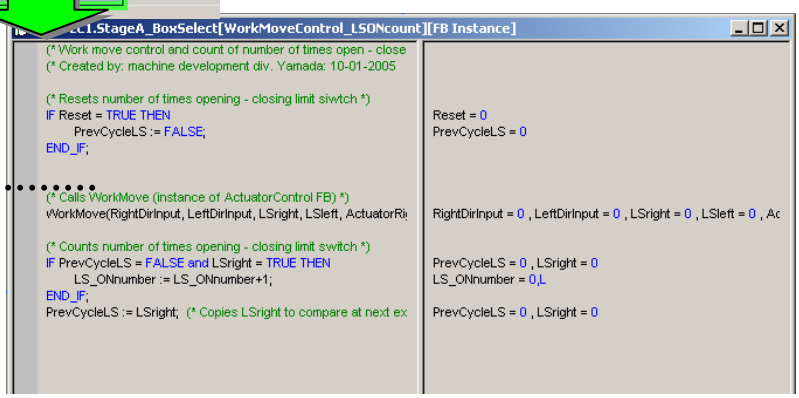


Passez du mode d'exécution au mode de surveillance.
Affichez l'instance FB « WorkMoveControl_LSONcount ».

Placez le curseur dans l'instruction d'appel FB, puis double-cliquez dessus ou cliquez sur le bouton .



Les valeurs actuelles des variables correspondant au programme sont surveillées dans l'instance ST FB (adresse attribuée).



Programme ST

Variables et valeurs actuelles

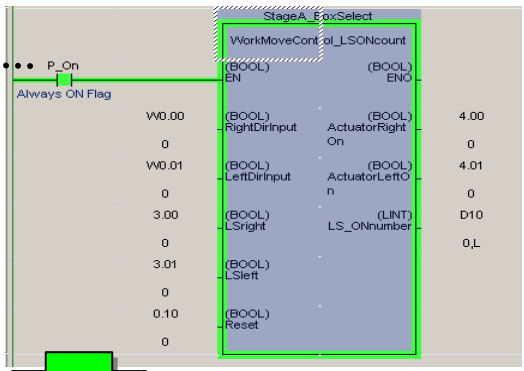
Définissez la valeur actuelle du paramètre d'instruction d'appel FB, puis confirmez la condition d'exécution. Définissez les conditions suivantes :

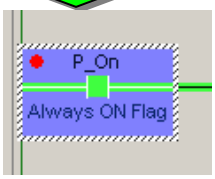
- RightDirInput : ON
- LeftDirInput : OFF
- LSright : OFF
- LSleft : ON
- Reset : OFF

Dans ce cas, les sorties suivantes sont attendues :

- ActuatorRightOn : ON
- ActuatorLeftOn : OFF
- LS_ONnumber : 1

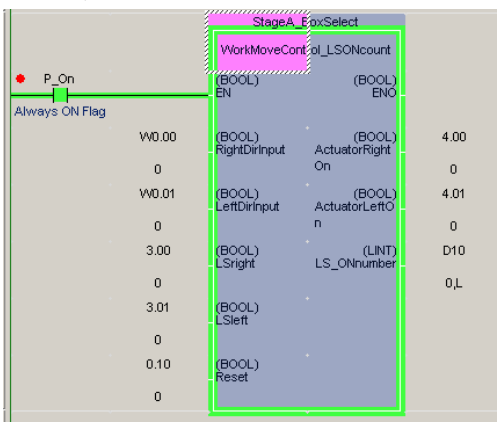
Placez le curseur sur l'entrée de gauche de l'instruction d'appel FB, puis cliquez sur le bouton .





Le programme s'arrête au point.

Cliquez sur .



Effectuez un contact d'entrée de point d'arrêt qui s'arrête à l'étape suivante de l'instruction d'appel FB.

@Appuyez sur [←] [↓][↓]

ENT

1

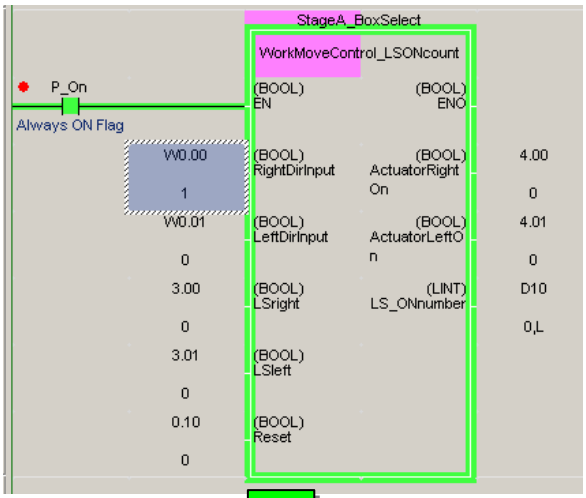
ENT

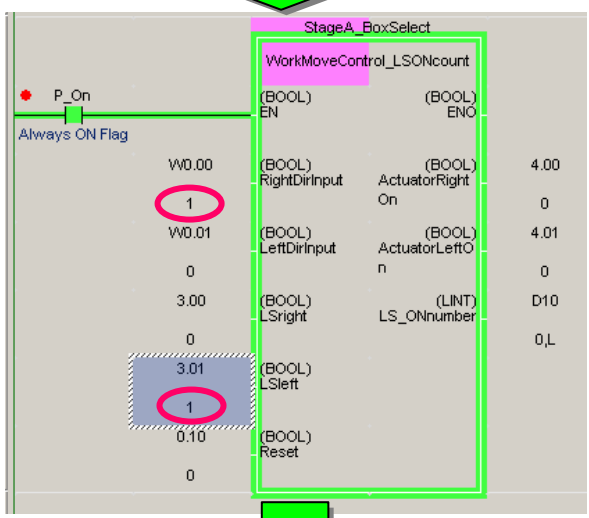
Appuyez sur [↓][↓][↓]

1

ENT

Activez les paramètres d'entrée « RightDirInput » et « LSleft » dans l'instruction d'appel FB.

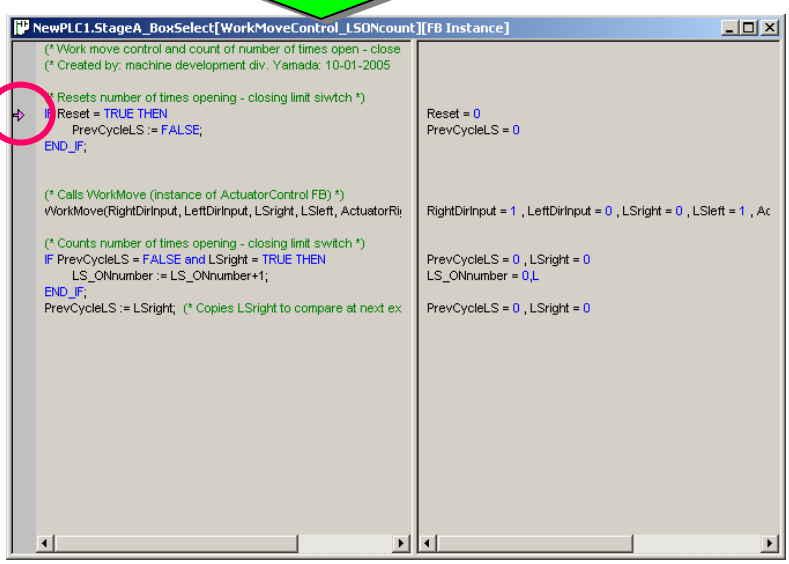




Les paramètres d'entrée nécessaires sont définis.

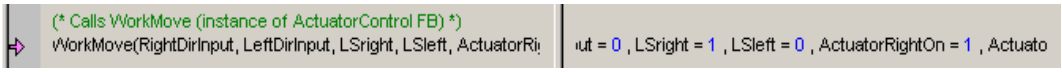
Cliquez sur .

Position de l'exécution de la surveillance ST

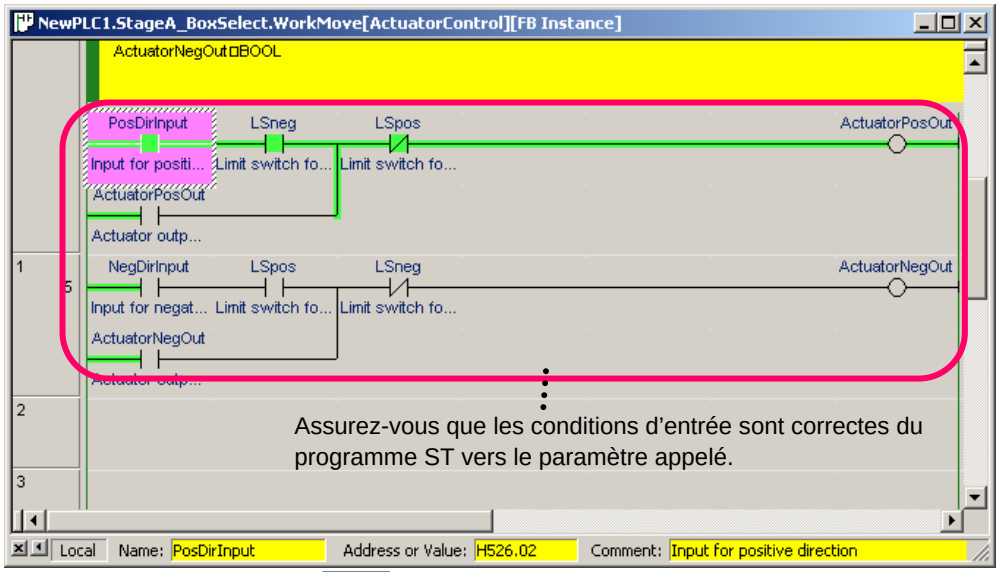


Le curseur se place sur la position de la première ligne du programme ST appelé.

Cliquez deux fois sur .

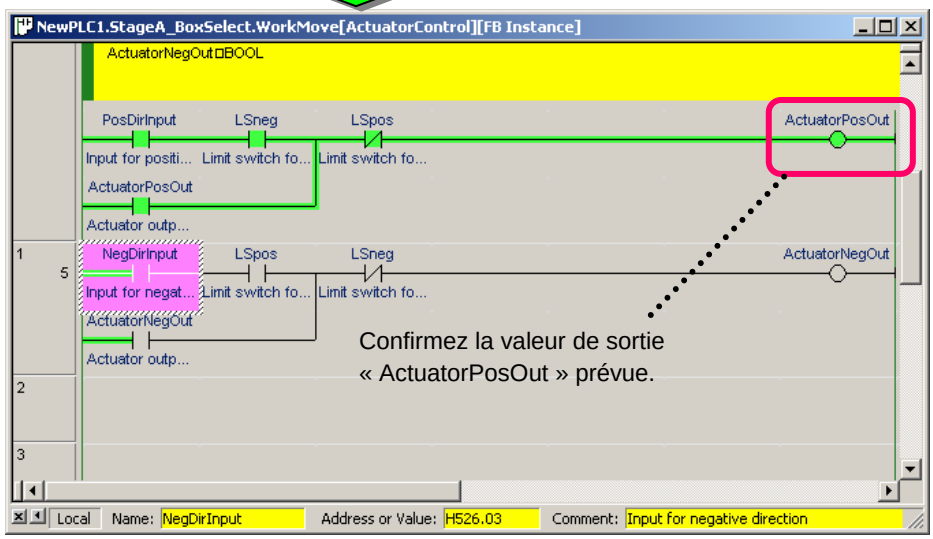


Transition du programme ST vers le programme contact FB appelé.



Assurez-vous que les conditions d'entrée sont correctes du programme ST vers le paramètre appelé.

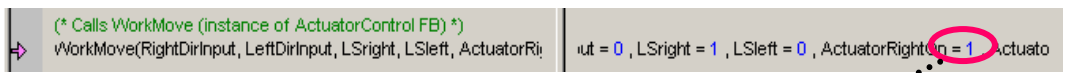
Cliquez cinq fois sur .



Confirmez la valeur de sortie « ActuatorPosOut » prévue.

Cliquez sur .

La confirmation est terminée.
Revenez dans le programme ST appelant.



Assurez-vous que le résultat de traitement du circuit précédent est correctement répercuté dans l'écran de surveillance du programme ST appelant.

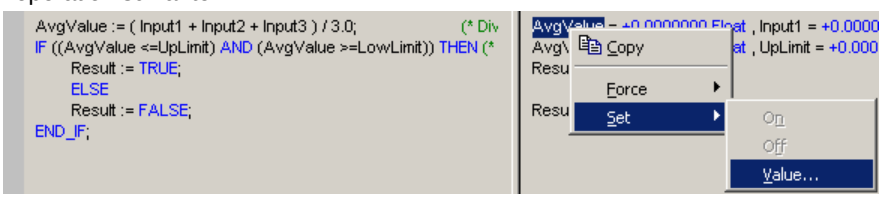
Cliquez sur  .

Transférez vers le programme appelant.

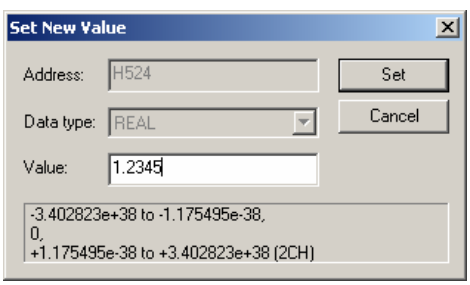
Assurez-vous que le paramètre de sortie est correctement répercuté.

Remarque

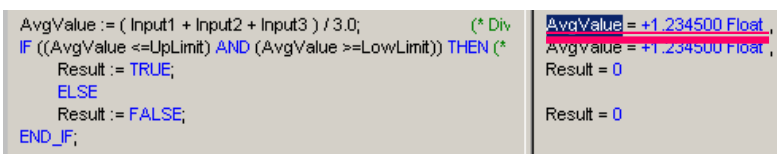
Il est possible de modifier la valeur actuelle du paramètre du programme ST à l'aide de l'opération suivante.



Placez le curseur sur le paramètre à modifier, cliquez dessus avec le bouton droit, puis sélectionnez Définir ⇒ Valeur.



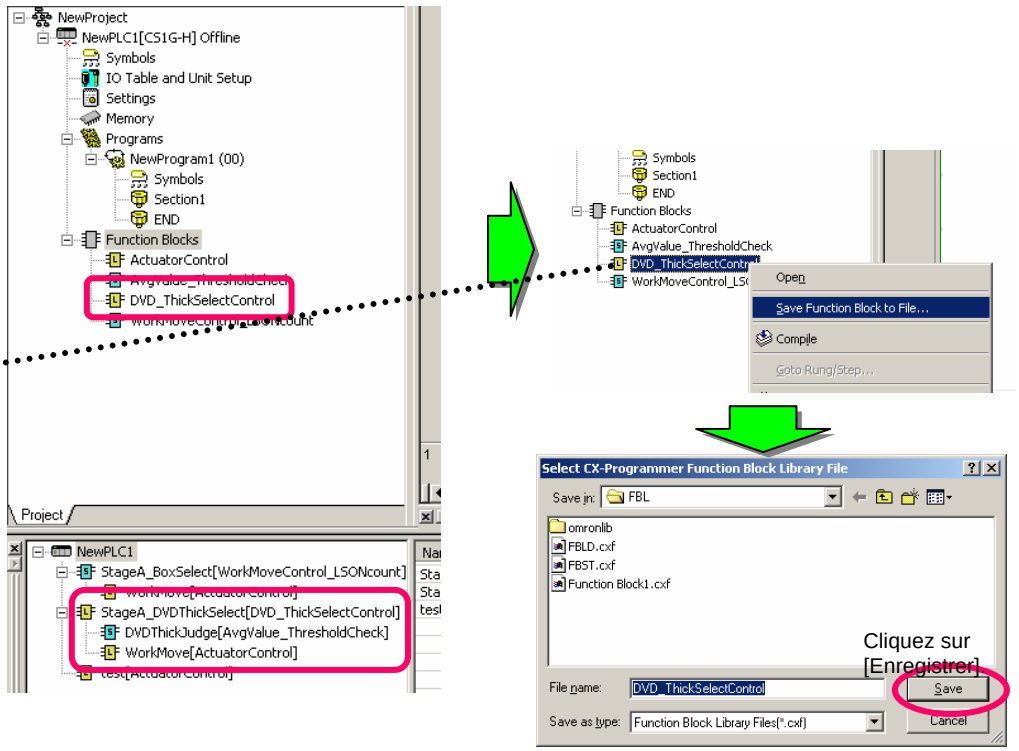
Définissez la valeur, puis cliquez sur [Définir].



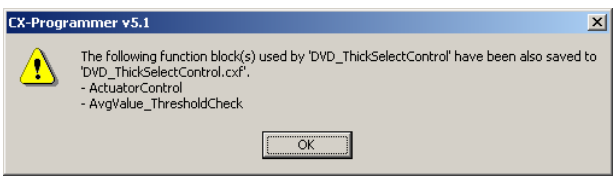
6. Création de la bibliothèque de définitions FB

Pour pouvoir réutiliser la définition de FB vérifiée, elle doit être incorporée dans une bibliothèque (fichier).
Vérifiez la hiérarchie à l'aide de l'espace de travail Projet et du visualiseur d'instance FB, puis déterminez la définition FB à incorporer dans la bibliothèque. Dans cet exemple, il s'agit du FB « DVD_ThickSelectControl ».

Sélectionnez le FB « DVD_ThickSelectControl », cliquez dessus avec le bouton droit, puis sélectionnez [Enregistrer bloc fonction dans le fichier] dans le menu contextuel.



Le dossier d'enregistrement par défaut est C:\Program Files\Omron\CX-One\FBL.
Il est possible de le modifier à l'aide de l'option CX-Programmer Dossier de stockage de la bibliothèque FB.
La bibliothèque FB OMRON est située dans le dossier omronlib.
Créez un dossier afin de faciliter le classement (Userlib\DVD, par exemple).



Lors de l'enregistrement d'une définition FB qui appelle un autre FB, les deux définitions FB sont enregistrées. Lors de l'extraction d'un projet, la relation appelant/appelé est préservée.
Il est plus simple de gérer la définition FB car une définition FB enregistrée est intégrée.

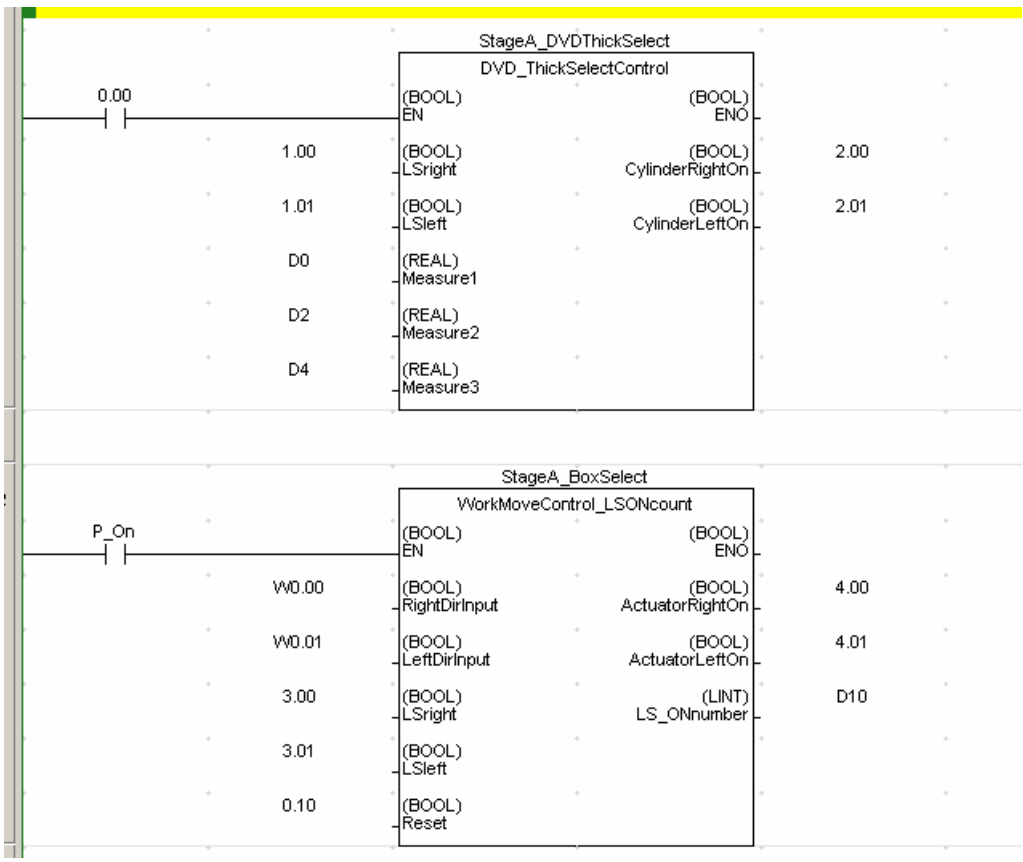
7. Entrée du programme principal

Ajoutez le programme principal dans un fichier de projet contenant la définition FB déboguée. Le programme à entrer correspond à celui décrit à la section 4-5. Description complète du programme, page 5-7.

[Variables globales]

Name	Data Type	Address / Value	Rack Location
StageA_BoxSelect	FB [WorkMoveControl_LSONcount]	N/A [Auto]	
StageA_DVDThickSelect	FB [DVD_ThickSelectControl]	N/A [Auto]	

* Les variables d'instance autres que celles utilisées pour FB sont omises.



Pour savoir comment entrer un programme, reportez-vous aux pages 2-6 à 2-9.

8. Débogage du programme principal

Le programme principal doit être débogué en tenant compte des éléments suivants :

- Zones de programme sans rapport avec les FB
- Zones de programme en rapport avec un paramètre d'entrée de FB
- Zones de programme faisant référence à un paramètre de sortie de FB

Les explications correspondantes sont omises car le programme principal de cet exemple ne contient aucune des zones ci-dessus.

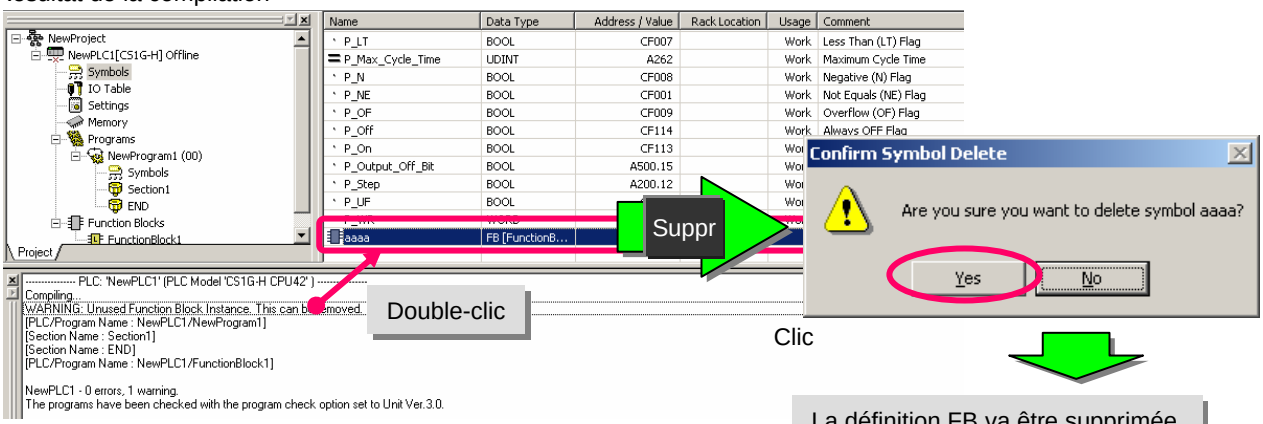
Informations supplémentaires

Suppression de définitions de bloc de fonctions non utilisées

Lors de la suppression de définitions de bloc de fonctions inutilisées, il ne suffit pas de supprimer l'instruction d'appel des blocs de fonctions. En effet, les définitions d'instance de bloc de fonctions sont enregistrées dans la table globale des symboles. Dans ce cas, une fois la compilation (contrôle du programme) terminée, les instances de bloc de fonctions inutilisées s'affichent dans la fenêtre Sortie. Vous pouvez identifier les définitions inutilisées et les supprimer en toute simplicité.
Les définitions et les instances de bloc de fonctions font partie d'un programme utilisateur dans l'UC, même si elles ne sont pas appelées. Par conséquent, il est recommandé de supprimer les définitions et les instances FB inutilisées avant de transférer le programme vers l'UC.

Exécutez la compilation F7

Résultat de la compilation



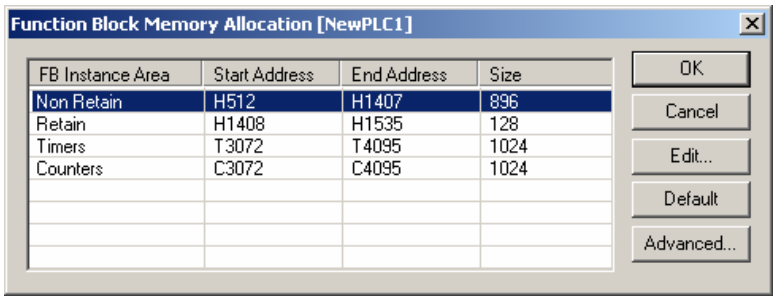
La définition FB va être supprimée.

Allocation de mémoire pour des blocs de fonctions

Il est nécessaire d'allouer la mémoire requise pour chaque instance FB afin d'exécuter les blocs de fonctions.
CX-Programmer alloue la mémoire automatiquement en fonction des informations de la boîte de dialogue des paramètres suivantes :
(menu API ® Mémoire bloc fonction ® Allocation de mémoire bloc de fonction)
Il existe 4 types de zones : Rejeter, Conserver, Temporisateurs et Compteurs. Modifiez les paramètres si nécessaire.

- Remarque applicable lors de la modification des paramètres
  Si vous modifiez la zone Rejeter ou Conserver, tenez compte des zones de mémoire allouée pour ls cartes E/S spéciales et les cartes réseau.

- Zone de mémoire spéciale pour les blocs de fonctions
  Les UC CS1/CJ1-H/CJ1M (version de carte : 3.0 ou ultérieure) disposent d'une zone de mémoire spéciale, à savoir la zone de relais de maintien (H) étendue.
  L'adresse de la zone va de H512 à H1535. CX-Programmer définit la zone par défaut.
  Notez que la zone ne peut pas être utilisée pour les opérandes de commande de schéma.



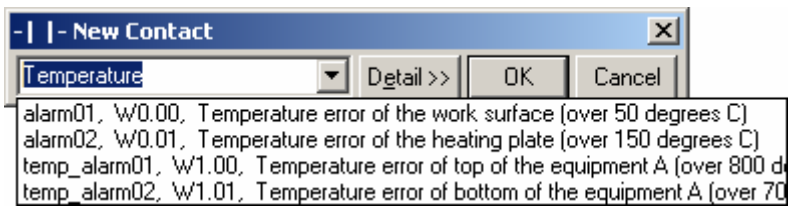
Fonctions utiles

Entrée d'opérande de commande - Recherche automatique et affichage de liste

Il est possible d'afficher automatiquement la liste des noms de symboles ou des commentaires E/S lors de l'entrée d'opérandes de commandes.
Lors de l'entrée de l'opérande de contact ou de sortie (ou instructions spéciales), entrez une chaîne. La liste déroulante est alors automatiquement mise à jour et affiche les noms des symboles ou les commentaires E/S en fonction de la chaîne définie. Sélectionnez un élément de la liste pour définir les informations sur l'opérande.
Cette méthode permet d'entrer de manière efficace des informations sur les symboles enregistrés dans le schéma de contact.

Exemple : Entrez le texte « Temperature » dans le champ de modification de la boîte de dialogue d'un opérande.

Cliquez sur ▼ ou appuyez sur [F4]. Tous les symboles ou adresses dont le commentaire E/S contient le texte « Temperature » s'affichent. Voir ci-dessous :



Par exemple, sélectionnez « temp_alarm01, W1.00, Temperature error of upper case of MachineA » dans la liste. L'opérande utilise alors le symbole « alarm01 ».

Fonction de protection des FB

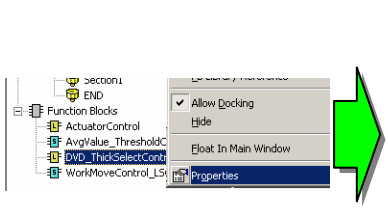
Il est possible de mettre en œuvre des mesures de prévention en définissant le mot de passe dans la définition de bloc de fonctions attribuée dans le fichier de projet. Il s'agit d'une protection concernant l'utilisation, les fuites de savoir-faire du programme et les modifications non autorisées.

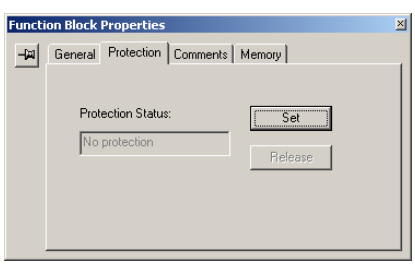
- Interdire l'écriture et l'affichage

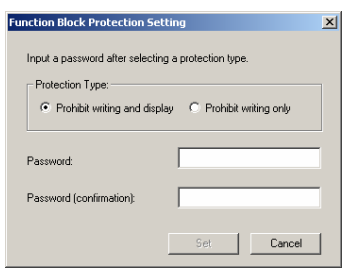
Si vous activez la classe de protection Interdire l'écriture et l'affichage, il est impossible d'afficher le contenu de la définition du bloc de fonctions correspondante. Activez la protection par mot de passe pour la définition du bloc de fonctions afin d'empêcher les fuites de savoir-faire du programme.

- Interdire l'écriture seule

Si vous activez la classe de protection Interdire l'écriture seule, il est impossible d'écrire ou de modifier le contenu de la définition du bloc de fonctions correspondante. Activez la protection par mot de passe pour la définition du bloc de fonctions afin d'empêcher toute modification interdite du programme.







Annexe. Exemples de texte structuré

Exemples d'instructions IF

IF *expression1* THEN *statement-list1*
[ELSIF *expression2* THEN *statement-list2*]
[ELSE *statement-list3*]
END_IF;

Les expressions expression1 et expression2 doivent se rapporter à une valeur booléenne. La liste d'instructions est une liste de plusieurs instructions simples telles que a:=a+1; b:=3+c; etc.

Le mot clé IF exécute liste d'instructions1 si expression1 est vraie ; si ELSIF est présent et expression1 est fausse et expression2 est vraie, il exécute liste d'instructions2 ; si ELSE est présent et expression1 ou expression2 est fausse, il exécute liste d'instructions3. Après l'exécution de liste d'instructions1, liste d'instructions2 ou de liste d'instructions3, la commande passe à l'instruction suivante après END_IF.

Une instruction IF peut contenir plusieurs instructions ELSIF, mais une seule instruction ELSE.

Les instructions IF peuvent être imbriquées dans d'autres instructions IF (voir exemple 5).

Exemple 1

```
IF a > 0 THEN
  b := 0;
END_IF;
```

Dans cet exemple, si la variable « a » est supérieure à zéro, la variable « b » reçoit la valeur zéro.

Si « a » n'est pas supérieure à zéro, aucune action n'est effectuée sur la variable « b » et la commande passe aux étapes de programme suivant la clause END_IF.

Exemple 2

```
IF a THEN
  b := 0;
END_IF;
```

Dans cet exemple, si la variable « a » est vraie, la variable « b » reçoit la valeur zéro.

Si « a » est fausse, aucune action n'est effectuée sur la variable « b » et la commande passe aux étapes de programme suivant la clause END_IF.

Exemple 3

```
IF a > 0 THEN
  b := TRUE;
ELSE
  b := FALSE;
END_IF;
```

Dans cet exemple, si la variable « a » est supérieure à zéro, la variable « b » reçoit la valeur TRUE (1) et la commande passe aux étapes de programme suivant la clause END_IF.

Si « a » n'est pas supérieure à zéro, aucune action n'est effectuée sur la variable « b », la commande passe aux étapes de programme suivant la clause END_IF et « b » reçoit la valeur FALSE (0).

La commande passe ensuite aux étapes de programme suivant la clause END_IF.

Exemple 4

```
IF a < 10 THEN
  b := TRUE;
  c := 100;
ELSIF a  > 20 THEN
  b := TRUE;
  c := 200;
ELSE
  b := FALSE;
  c := 300;
END_IF;
```

Dans cet exemple, si la variable « a » est inférieure à 10, la variable « b » reçoit la valeur TRUE (1) et la variable « c » reçoit la valeur 100. La commande passe ensuite aux étapes de programme suivant la clause END_IF.

Si la variable « a » est supérieure ou égale à 10, la commande passe aux étapes de programme suivant la clause ELSE_IF et si la variable « a » est supérieure à 20, la variable « b » reçoit la valeur TRUE (1) et la variable « c » reçoit la valeur 200. La commande passe ensuite aux étapes de programme suivant la clause END_IF.

Si la variable « a » est entre 10 et 20 (donc les deux conditions précédentes IF et ELSE_IF sont fausses), la commande passe aux étapes de programme suivant la clause ELSE, la variable « b » reçoit la valeur FALSE (0) et la variable « c » reçoit la valeur 300. La commande passe ensuite aux étapes de programme suivant la clause END_IF.

Exemples d'instructions IF

Exemple 5

```
IF a THEN
  b :=  TRUE;
ELSE
  IF c>0 THEN
    d := 0;
  ELSE
    d := 100;
  END_IF;
  d := 400;
END_IF;
```

Dans cet exemple (exemple d'instruction IF … THEN imbriquée), si la variable « a » est vraie (1), la variable « b » reçoit la valeur TRUE (1) et la commande passe aux étapes de programme suivant la clause END_IF.

Si « a » est fausse (0), aucune action n'est effectuée sur la variable « b », la commande passe aux étapes de programme suivant la clause ELSE (ici, une autre instruction IF ... THEN, qui est exécutée comme décrit à l'exemple 3, bien que toutes les instructions IEC61131-3 prises en charge puissent être utilisées).

Après l'exécution de l'instruction IF ... THEN décrite, la variable « d » reçoit la valeur 400.

La commande passe ensuite aux étapes de programme suivant la clause END_IF.

Exemples d'instructions WHILE

WHILE *expression* DO
** *statement-list*;**
END_WHILE;

L'expression WHILE doit se rapporter à une valeur booléenne. La liste d'instructions est une liste de plusieurs instructions simples.

Le mot clé WHILE exécute plusieurs fois la liste d'instructions tant que l'expression est vraie. Lorsque l'expression devient fausse, la commande passe à l'instruction suivant immédiatement END_WHILE.

Exemple 1

```
WHILE a < 10 DO
  a := a + 1;
  b := b * 2.0;
END_WHILE;
```

Dans cet exemple, l'expression WHILE est évaluée et si elle est vraie (si la variable « a » est inférieure à 10), la liste d'instructions (a := a + 1; et b := b * 2.0;) est exécutée. Après l'exécution de la liste d'instructions, la commande repasse au début de l'expression WHILE. Ce processus est répété tant que la variable « a » est inférieure à 10. Lorsqu'elle est supérieure ou égale à 10, la liste d'instructions ne s'exécute pas et la commande passe aux étapes de programme suivant la clause END_WHILE.

Exemple 2

```
WHILE a DO
  b := b + 1;
  IF b > 10 THEN
    a:= FALSE;
  END_IF;
END_WHILE;
```

Dans cet exemple, l'expression WHILE est évaluée et si elle est vraie (si la variable « a » est vraie), la liste d'instructions (b := b + 1; et l'instruction IF ... THEN) est exécutée. Après l'exécution de la liste d'instructions, la commande repasse au début de l'expression WHILE. Ce processus se répète tant que la variable « a » est vraie. Lorsque la variable « a » est fausse, la liste d'instructions ne s'exécute pas et la commande passe aux étapes de programme suivant la clause END_WHILE.

Exemple 3

```
WHILE (a + 1) >= (b * 2) DO
  a := a + 1;
  b := b / c;
END_WHILE;
```

Dans cet exemple, l'expression WHILE est évaluée et si elle est vraie (si la variable « a » plus 1 donne une valeur supérieure ou égale à la variable « b » multipliée par 2), la liste d'instructions (a := a + 1; et b := b / c;) est exécutée. Après l'exécution de la liste d'instructions, la commande repasse au début de l'expression WHILE. Ce processus se répète tant que l'expression WHILE est vraie. Lorsque l'expression WHILE est fausse, la liste d'instructions ne s'exécute pas et la commande passe aux étapes de programme suivant la clause END_WHILE.

Exemples d'instructions WHILE

Exemple 4

```
WHILE (a - b) <= (b + c) DO
  a := a + 1;
  b := b * a;
END_WHILE;
```

Dans cet exemple, l'expression WHILE est évaluée et si elle est vraie (si la variable « a » moins la variable « b » donne une valeur inférieure ou égale à la variable « b » plus la variable « c »), la liste d'instructions (a := a + 1; et b := b * a;) est exécutée. Après l'exécution de la liste d'instructions, la commande repasse au début de l'expression WHILE. Ce processus se répète tant que l'expression WHILE est vraie. Lorsque l'expression WHILE est fausse, la liste d'instructions ne s'exécute pas et la commande passe aux étapes de programme suivant la clause END_WHILE.

Exemples d'instructions REPEAT

REPEAT
  ***statement-list*;**
UNTIL *expression*
END_REPEAT;

L'expression REPEAT doit se rapporter à une valeur booléenne. La liste d'instructions est une liste de plusieurs instructions simples.

Le mot clé REPEAT exécute plusieurs fois la liste d'instructions tant que l'expression est fausse. Lorsque l'expression devient vraie, la commande passe à l'instruction suivant immédiatement END_REPEAT.

Exemple 1

```
REPEAT
  a := a + 1;
  b := b * 2.0;
UNTIL a > 10
END_REPEAT;
```

Dans cet exemple, la liste d'instructions (a := a + 1; et b := b * 2.0;) est exécutée. Après l'exécution de la liste d'instructions, l'expression UNTIL est évaluée ; si elle est fausse (la variable « a » est inférieure ou égale à 10), la commande repasse au début de l'expression REPEAT et la liste d'instructions est à nouveau exécutée. Ce processus se répète tant que l'expression UNTIL est fausse. Lorsque l'expression UNTIL est vraie (la variable « a » est inférieure à 10), la commande passe aux étapes de programme suivant la clause END_REPEAT.

Exemple 2

```
REPEAT
  b := b + 1;
  IF b > 10 THEN
    a:= FALSE;
  END_IF;
UNTIL a
END_REPEAT;
```

Dans cet exemple, la liste d'instructions (b := b + 1; et l'instruction IF ... THEN) est exécutée. Après l'exécution de la liste d'instructions, l'expression UNTIL est évaluée ; si elle est fausse (la variable « a » est fausse), la commande repasse au début de l'expression REPEAT et la liste d'instructions est à nouveau exécutée. Ce processus se répète tant que l'expression UNTIL est fausse. Lorsque l'expression UNTIL est vraie (la variable « a » est vraie), la commande passe aux étapes de programme suivant la clause END_REPEAT.

Exemple 3

```
REPEAT
  a := a + 1;
  b := b / c;
UNTIL (a + 1) >= (b * 2)
END_REPEAT;
```

Dans cet exemple, la liste d'instructions (a := a + 1; et b := b / c;) est exécutée. Après l'exécution de la liste d'instructions, l'expression UNTIL est évaluée ; si elle est fausse (la variable « a » plus 1 donne une valeur inférieure à la variable « b » multipliée par 2), la commande repasse au début de l'expression REPEAT et la liste d'instructions est à nouveau exécutée. Ce processus se répète tant que l'expression UNTIL est fausse. Lorsque l'expression UNTIL est vraie (la variable « a » plus 1 donne une valeur supérieure ou égale à la variable « b » multipliée par 2), la commande passe aux étapes de programme suivant la clause END_REPEAT.

Exemple 4

```
REPEAT
  a := a + 1;
  b := b * a;
UNTIL (a - b) <= (b + c)
END_REPEAT;
```

Dans cet exemple, la liste d'instructions (a := a + 1; et b := b * a;) est exécutée. Après l'exécution de la liste d'instructions, l'expression UNTIL est évaluée ; si elle est fausse (la variable « a » moins la variable « b » donne une valeur supérieure à la variable « b » plus la variable « c »), la commande repasse au début de l'expression REPEAT et la liste d'instructions est à nouveau exécutée. Ce processus se répète tant que l'expression UNTIL est fausse. Lorsque l'expression UNTIL est vraie (la variable « a » moins la variable « b » donne une valeur inférieure ou égale à la variable « b » plus la variable « c »), la commande passe aux étapes de programme suivant la clause END_REPEAT.

Exemples d'instructions FOR

FOR *control variable* := *integer expression1* TO *integer expression2* [BY *integer expression3*] DO

liste d'instructions;

END_FOR;

La variable de commande FOR doit être une variable de type Integer. Les expressions Integer FOR doivent se rapporter au même type de variable Integer que la variable de commande. La liste d'instructions est une liste de plusieurs instructions simples.

Le mot clé FOR exécute plusieurs fois la liste d'instructions tant que la variable de commande se trouve dans la plage comprise entre expression1 Integer et expression2 Integer. Si le mot clé BY est présent, la variable de commande est incrémentée de expression3 Integer, sinon elle est incrémentée de un par défaut. La variable de commande est incrémentée après chaque appel de la liste d'instructions. Lorsque la variable de commande ne se trouve plus dans la plage comprise entre expression1 Integer et expression2 Integer, la commande passe à l'instruction suivant immédiatement END_FOR.

Les instructions FOR peuvent être imbriquées dans d'autres instructions FOR.

Example 1

```
FOR a := 1 TO 10 DO
  b := b + a;
END_FOR;
```

Dans cet exemple, l'expression FOR est évaluée initialement et la variable « a » est initialisée avec la valeur 1. La valeur de la variable « a » est ensuite comparée avec la valeur « TO » et, si elle est inférieure ou égale à 10, la liste d'instructions (b := b + a;) est exécutée. La variable « a » est ensuite incrémentée de 1 et la commande repasse au début de l'instruction FOR. La variable « a » est à nouveau comparée à la valeur « TO » et, si elle est inférieure ou égale à 10, la liste d'instructions est à nouveau exécutée. Ce processus se répète jusqu'à ce que la valeur de la variable « a » soit supérieure à 10 puis la commande passe aux étapes de programme suivant la clause END_FOR.

Example 2

```
FOR a := 1 TO 10 BY 2 DO
  b := b + a;
  c := c + 1.0;
END_FOR;
```

Dans cet exemple, l'expression FOR est évaluée initialement et la variable « a » est initialisée avec la valeur 1. La valeur de la variable « a » est ensuite comparée avec la valeur « TO » et, si elle est inférieure ou égale à 10, la liste d'instructions (b := b + a; et c := c + 1.0;) est exécutée. La variable « a » est ensuite incrémentée de 2 et la commande repasse au début de l'instruction FOR. La variable « a » est à nouveau comparée à la valeur « TO » et, si elle est inférieure ou égale à 10, la liste d'instructions est à nouveau exécutée. Ce processus se répète jusqu'à ce que la valeur de la variable « a » soit supérieure à 10 puis la commande passe aux étapes de programme suivant la clause END_FOR.

Example 3

```
FOR a := 10 TO 1 BY -1 DO
  b := b + a;
  c := c + 1.0;
END_FOR;
```

Dans cet exemple, l'expression FOR est évaluée initialement et la variable « a » est initialisée avec la valeur 10. La valeur de la variable « a » est ensuite comparée avec la valeur « TO » et, si elle est supérieure ou égale à 1, la liste d'instructions (b := b + a; et c := c + 1.0;) est exécutée. La variable « a » est ensuite décrémentée de 1 et la commande repasse au début de l'instruction FOR. La variable « a » est à nouveau comparée à la valeur « TO » et, si elle est supérieure ou égale à 1, la liste d'instructions est à nouveau exécutée. Ce processus se répète jusqu'à ce que la valeur de la variable « a » soit inférieure à 1, puis la commande passe aux étapes de programme suivant la clause END_FOR.

Example 4

```
FOR a := b + 1 TO c + 2 DO
  d := d + a;
  e := e + 1;
END_FOR;
```

Dans cet exemple, l'expression FOR est évaluée initialement et la variable « a » est initialisée avec la valeur de la variable « b » plus 1. La valeur « TO » de l'instruction FOR est évaluée par rapport à la valeur de la variable « c » plus 2. La valeur de la variable « a » est ensuite comparée à la valeur « TO » et, si elle est inférieure ou égale à celle-ci, la liste d'instructions (d := d + a; et e := e +1;) est exécutée. La variable « a » est ensuite incrémentée de 1 et la commande repasse au début de l'instruction FOR. La variable « a » est à nouveau comparée à la valeur « TO » et, si elle est inférieure ou égale à celle-ci, la liste d'instructions est à nouveau exécutée. Ce processus se répète jusqu'à ce que la valeur de la variable « a » soit supérieur à la valeur « TO », puis la commande passe aux étapes de programme suivant la clause END_FOR.

Exemples d'instructions FOR

Exemple 5

```
FOR a := b + c TO d - e BY f DO
  g := g + a;
  h := h + 1.0;
END_FOR;
```

Dans cet exemple, l'expression FOR est évaluée initialement et la variable « a » est initialisée avec la valeur de la variable « b » plus la variable « c ». La valeur « TO » de l'instruction FOR est évaluée par rapport à la valeur de la variable « c » moins la variable « d ». La valeur de la variable « a » est ensuite comparée à la valeur « TO ». Si la valeur de la variable « f » est positive et que la valeur de la variable « a » est inférieure ou égale à la valeur « TO », la liste d'instructions (g := g + a; et h := h +1.0;) est exécutée. Si la valeur de la variable « f » est négative et que la valeur de la variable « a » est supérieure ou égale à la valeur « TO », la liste d'instructions (g := g + a; et h := h +1.0;) est également exécutée. La variable « a » est ensuite incrémentée ou décrémentée de la valeur de la variable « f » et la commande repasse au début de l'instruction FOR. La variable « a » est à nouveau comparée à la valeur « TO » et la liste d'instructions est exécutée le cas échéant (comme décrit ci-dessus).

Ce processus se répète jusqu'à ce que la valeur de la variable « a » soit supérieure à la valeur « TO » (si la valeur de la variable « f » est positive) ou jusqu'à ce que la valeur de la variable « a » soit inférieure à la valeur « TO » (si la valeur de la variable « f » est négative), puis la commande passe aux étapes de programme suivant la clause END_FOR.

Exemples d'instructions CASE

CASE ***expression*** **OF**
étiquette case1 [, étiquette case2] [.. étiquette case3] : liste d'instructions1;
[ELSE
liste d'instructions2]
END_CASE;

L'expression CASE doit se rapporter à une valeur entière. La liste d'instructions est une liste de plusieurs instructions simples. Les étiquettes de case doivent être des valeurs entières littérales valides, comme 0, 1, +100, -2 etc.

Le mot clé CASE évalue l'expression et exécute la liste d'instructions associée à une étiquette case dont la valeur correspond à l'expression. La commande passe à l'instruction suivant immédiatement END_CASE. Si aucune correspondance n'est trouvée dans les étiquettes case précédentes et qu'une commande ELSE est présente, la liste d'instructions associée au mot clé ELSE est exécutée. S'il n'y a pas de mot clé ELSE, la commande passe à l'instruction suivant immédiatement END_CASE.

Une instruction CASE peut contenir plusieurs étiquettes case différentes (avec les listes d'instructions associées), mais une seule instruction ELSE.

L'opérateur « , » sert à lister plusieurs étiquettes case associées à une même liste d'instructions.

L'opérateur « .. » marque une plage d'étiquettes case. Si l'expression CASE se trouve dans cette plage, la liste d'instructions qui lui est associée est exécutée, par exemple l'étiquette case 1...10 : a:=a+1; exécute a:=a+1 si l'expression CASE est supérieure ou égale à 1 et inférieure à 10.

Exemple 1

```
CASE a OF
  2 :   b := 1;
  5 :   c := 1.0;
END_CASE;
```

Dans cet exemple, l'instruction CASE est évaluée puis comparée à chaque valeur de comparaison de l'instruction CASE (2 et 5 dans cet exemple).

Si la valeur de la variable « a » est égale à 2, cette liste d'instructions est exécutée (b := 1;). La commande passe ensuite aux étapes de programme suivant la clause END_CASE.

Si la valeur de la variable « a » est égale à 5, cette liste d'instructions est exécutée (c := 1.0;). La commande passe ensuite aux étapes de programme suivant la clause END_CASE.

Si la valeur de la variable « a » ne correspond à aucune valeur de comparaison de l'instruction CASE, la commande passe aux étapes de programme suivant la clause END_CASE.

Exemple 2

```
CASE a + 2 OF
  -2 :  b := 1;
  5 :   c := 1.0;
ELSE
  d := 1.0;
END_CASE;
```

Dans cet exemple, l'instruction CASE est évaluée puis comparée à chaque valeur de comparaison de l'instruction CASE (-2 et 5 dans cet exemple).

Si la valeur de la variable « a » plus 2 est égale à -2, cette liste d'instructions est exécutée (b := 1;). La commande passe ensuite aux étapes de programme suivant la clause END_CASE. Si la valeur de la variable « a » plus 2 est égale à 5, cette liste d'instructions est exécutée (c := 1.0;). La commande passe ensuite aux étapes de programme suivant la clause END_CASE. Si la valeur de la variable « a » plus 2 est différente de -2 et 5, cette liste d'instructions dans la condition ELSE est exécutée (d := 1.0;). La commande passe ensuite aux étapes de programme suivant la clause END_CASE.

Exemples d'instructions CASE

Exemple 3

```
 CASE  a + 3 * b OF
   1, 3 :    b := 2;
   7, 11 :  c := 3.0;
ELSE
   d := 4.0;
END_CASE;
```

Dans cet exemple, l'instruction CASE est évaluée puis comparée à chaque valeur de comparaison de l'instruction CASE (c'est-à-dire 1 ou 3 et 7 ou 11 dans cet exemple).

Si la valeur de la variable « a » plus 3 multipliée par la variable « b » est égale à 1 ou 3, cette liste d'instructions est exécutée (b := 2;). La commande passe ensuite aux étapes de programme suivant la clause END_CASE.

Si la valeur de la variable « a » plus 3 multipliée par la variable « b » est égale à 7 ou 11, cette liste d'instructions est exécutée (c := 3.0;). La commande passe ensuite aux étapes de programme suivant la clause END_CASE.

Si la valeur de la variable « a » plus 3 multipliée par la variable « b » est différente de 1, 3, 7 et 11, cette liste d'instructions dans la condition ELSE est exécutée (d := 4.0;). La commande passe ensuite aux étapes de programme suivant la clause END_CASE.

Exemple 4

```
 CASE  a OF
-2, 2, 4 :      b := 2;
                c := 1.0;
6..11, 13 :     c := 2.0;
1, 3, 5 :       c := 3.0;
ELSE
   b := 1;
   c := 4.0;
END_CASE;
```

Dans cet exemple, l'instruction CASE est évaluée puis comparée à chaque valeur de comparaison de l'instruction CASE, c'est-à-dire (-2, 2 ou 4) et (6 à 11 ou 13) et (1, 3 ou 5) dans cet exemple.

Si la valeur de la variable « a » est égale à -2, 2 ou 4, cette liste d'instructions est exécutée (b := 2; et c := 1.0;). La commande passe ensuite aux étapes de programme suivant la clause END_CASE.

Si la valeur de la variable « a » est égale à 6, 7, 8, 9, 10, 11 ou 13, cette liste d'instructions est exécutée (c := 2.0;). La commande passe ensuite aux étapes de programme suivant la clause END_CASE.

Si la valeur de la variable « a » est égale à 1, 3 ou 5, cette liste d'instructions est exécutée (c := 3.0;). La commande passe ensuite aux étapes de programme suivant la clause END_CASE.

Si la valeur de la variable « a » est différente de toutes les valeurs indiquées plus haut, cette liste d'instructions dans la condition ELSE est exécutée (b := 1; et c := 4.0;). La commande passe ensuite aux étapes de programme suivant la clause END_CASE.

Exemples d'instructions EXIT

WHILE ***expression*** **DO**
  liste d'instructions1;
  EXIT;
END_WHILE;
liste d'instructions2;

REPEAT
  liste d'instructions1;
  EXIT;
UNTIL ***expression***
END_REPEAT;
liste d'instructions2;

FOR ***variable de commande := expression1 Integer*** **TO** ***expression2 Integer*** **[BY** ***expression3 Integer*** **] DO**
  liste d'instructions1;
  EXIT;
END_FOR;
liste d'instructions2;

La *liste d'instructions* est une liste de plusieurs instructions simples.

Le mot clé EXIT interrompt l'exécution de la boucle répétitive pour passer à l'instruction suivante et ne peut être utilisée que dans les instructions répétitives (WHILE, REPEAT, FOR). Lorsque le mot clé EXIT est exécuté après la *liste d'instructions 1* dans la boucle répétitive, la commande passe immédiatement à la *liste d'instructions 2*.

Exemple 1

```
WHILE a DO
 IF c = TRUE THEN
   b:=0;EXIT;
 END_IF;
 IF b > 10 THEN
    a:= FALSE;
  END_IF;
END_WHILE;
d:=1;
```

Si la première expression IF est vraie (la variable « c » est vraie), la liste d'instructions (b := 0; et EXIT;) est exécutée pendant l'exécution de la boucle WHILE. Après l'exécution du mot clé EXIT, la boucle WHILE est interrompue et la commande passe à l'instruction suivante (d := 1) après la clause END_WHILE.

Exemple 2

```
 a:=FALSE;
FOR i:=1 TO 20 DO
    FOR j:=0 TO 9 DO
        IF i>=10 THEN
            n:=i*10+j;
            a:=TRUE;EXIT;
        END_IF;
    END_FOR;
    IF a THEN EXIT; END_IF;
END_FOR;
d:=1;
```

Si la première expression IF est vraie (i>=10 est vrai) dans la boucle interne FOR, la liste d'instructions (n:=i*10+j; et a:=TRUE; et EXIT;) est exécutée pendant l'exécution de la boucle FOR. Après l'exécution du mot clé EXIT, la boucle interne FOR est interrompue et la commande passe à l'instruction IF suivante après la clause END_FOR. Si cette expression IF est vraie (la variable « a » est vraie), le mot clé EXIT est exécuté, la boucle externe FOR est interrompue après la clause END_FOR et la commande passe à l'instruction suivante (d := 1).

Exemples d'instructions RETURN

liste d'instructions1;
RETURN;
liste d'instructions2;

La liste d'instructions est une liste de plusieurs instructions simples.

Le mot clé RETURN interrompt l'exécution à l'intérieur du bloc de fonctions après la liste d'instructions 1, puis la commande revient au programme qui appelle le bloc de fonctions sans exécuter la liste d'instructions 2.

Exemple 1

```
IF a_1*b>100 THEN
  c:=TRUE;RETURN;
END_IF;
IF a_2*(b+10)>100 THEN
  c:=TRUE;RETURN;
END_IF;
IF a_3*(b+20)>100 THEN
  c:=TRUE;
END_IF;
```

Si la première ou la deuxième instruction IF est vraie (« a_1 * b » est supérieur à 100 ou « a_2 * (b + 10) » supérieur à 100), l'instruction (c := TRUE; et RETURN;) est exécutée. L'exécution du mot clé RETURN interrompt l'exécution à l'intérieur du bloc de fonctions et la commande revient au programme qui appelle le bloc de fonctions.

Exemples de tableaux

nom de variable [index d'indice inférieur]

Un tableau est un ensemble de variables semblables. Vous pouvez définir la taille d'un tableau dans la table des variables des blocs de fonctions.

Il est possible d'accéder à une variable spécifique à l'aide de l'opérateur d'indice inférieur du tableau [ ].

L'index d'indice inférieur permet d'accéder à une variable particulière à l'intérieur d'un tableau. L'index d'indice inférieur doit être une valeur littérale positive, une expression Integer ou une variable entière. L'index d'indice inférieur est de base zéro. Une valeur d'index d'indice inférieur égale à zéro permet d'accéder à la première variable, une valeur d'index d'indice inférieur égale à un permet d'accéder à la deuxième variable, etc.

Avertissement

Si l'index d'indice inférieur est une expression Integer ou une variable entière, assurez-vous que la valeur de l'index d'indice inférieur résultante est dans une plage d'index valide du tableau. Evitez d'accéder à un tableau avec un index non valide. Reportez-vous à l'exemple 5 pour plus d'informations sur l'écriture d'un code sûr en cas d'utilisation d'offsets de tableaux de variables.

Exemple 1

```
a[0] := 1;
a[1] := -2;
a[2] : = 1+2;
a[3] : = b;
a[4] : = b+1;
```

Dans cet exemple, la variable « a » est un tableau de 5 éléments et son type de données est INT. Le type de données de la variable « b » est aussi INT. Lors de l'exécution, le premier élément du tableau est défini sur la valeur 1, le deuxième élément sur -2, le troisième élément sur 3 (1 + 2), le quatrième sur la valeur de la variable « b » et le cinquième sur la valeur de la variable « b » plus 1.

Exemple 2

```
c[0] := FALSE;
c[1] := 2>3;
```

Dans cet exemple, la variable « c » est un tableau de 2 éléments et son type de données est BOOL. Lors de l'exécution, le premier élément du tableau est défini sur FALSE et le deuxième élément sur FALSE, c'est-à-dire que 2 supérieur à 3 est évalué sur FALSE.

Exemples de tableaux

Exemple 3

```
d[9]:= 2.0;
```

Dans cet exemple, la variable « d » est un tableau de 10 éléments et son type de données est REAL. Lors de l'exécution, le dernier élément du tableau (le dixième) est défini sur 2.0.

Exemple 4

```
a[1] := b[2];
```

Dans cet exemple, la variable « a » et la variable « b » sont des tableaux du même type de données. Lors de l'exécution, la valeur du deuxième élément de la variable « a » est définie sur la valeur du troisième élément dans la variable « b ».

Exemple 5

```
a[b] := 1;
a[b+1] := 1;
a[(b+c) *( d-e)] := 1;
```

Remarque : dans la mesure où les variables et les expressions de type Integer sont utilisées pour accéder au tableau, la valeur d'index réelle n'est pas connue avant l'exécution. Vous devez donc vous assurer que l'index se trouve dans la plage valide du tableau « a ». Par exemple, contrôler la validité de l'index de tableau serait une méthode plus sûre :

```
f := (b+c) *( d-e);
IF (f >0) AND (f<5) THEN
   a[f] := 1;
END_IF;
```

Où le type de données de la variable « f » est INT.

Exemple 6

```
a[b[1]]:= c;
a[b[2] + 3]:= c;
```

Cet exemple démontre comment une expression d'élément de tableau peut être utilisée à l'intérieur d'une autre expression d'élément de tableau.

Fonctions numériques et fonctions arithmétiques

Fonction	Nom	Type de données d'argument	Type de valeur renvoyée	Opération	Exemple
ABS(argument)	Valeur absolue	INT, DINT, LINT, UINT, UDINT, ULINT, REAL, LREAL	NT, DINT, LINT, UINT, UDINT, ULINT, REAL, LREAL	| argument |	a:=ABS(b)
SQRT(argument)	Racine carrée	REAL, LREAL	REAL, LREAL		a:=SQRT(b)
LN(argument)	Logarithme naturel	REAL, LREAL	REAL, LREAL		a:=LN(b)
LOG(argument)	Logarithme commun	REAL, LREAL	REAL, LREAL		a:=LOG(b)
EXP(argument)	Exponentiel naturel	REAL, LREAL	REAL, LREAL		a:=EXP(b)
SIN(argument)	Sinus	REAL, LREAL	REAL, LREAL	SIN(argument)	a:=SIN(b)
COS(argument)	Cosinus	REAL, LREAL	REAL, LREAL	COS(argument)	a:=COS(b)
TAN(argument)	Tangente	REAL, LREAL	REAL, LREAL	TAN(argument)	a:=TAN(b)
ASIN(argument)	Arc sinus	REAL, LREAL	REAL, LREAL		a:=ASIN(b)
ACOS(argument)	Arc cosinus	REAL, LREAL	REAL, LREAL		a:=ACOS(b)
ATAN(argument)	Arc tangente	REAL, LREAL	REAL, LREAL		a:=ATAN(b)
EXPT(base, exposant)	Exponentiel	Base : REAL, LREAL Exposant : INT, DINT, LINT, UINT, UDINT, ULINT	REAL, LREAL		a:=EXPT(b, c)