Unity Pro

Langages de programmation et structure
Manuel de référence

05/2010

35006145.09

www.schneider-electric.com

Schneider Electric

Le présent document comprend des descriptions générales et/ou des caractéristiques techniques générales sur la performance des produits auxquels il se réfère. Le présent document ne peut être utilisé pour déterminer l'aptitude ou la fiabilité de ces produits pour des applications utilisateur spécifiques et n'est pas destiné à se substituer à cette détermination. Il appartient à chaque utilisateur ou intégrateur de réaliser, sous sa propre responsabilité, l'analyse de risques complète et appropriée, et d'évaluer et de tester les produits dans le contexte de leur l'application ou utilisation spécifique. Ni la société Schneider Electric, ni aucune de ses filiales ou sociétés dans lesquelles elle détient une participation, ne peut être tenue pour responsable de la mauvaise utilisation des informations contenues dans le présent document. Si vous avez des suggestions, des améliorations ou des corrections à apporter à cette publication, veuillez nous en informer.

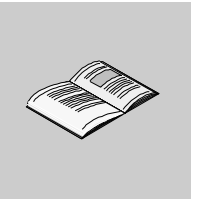
Toutes les réglementations locales, régionales et nationales en matière de sécurité doivent être respectées lors de l'installation et de l'utilisation de ce produit. Pour des raisons de sécurité et afin de garantir la conformité aux données système documentées, seul le fabricant est habilité à effectuer des réparations sur les composants.

Lorsque des équipements sont utilisés pour des applications présentant des exigences de sécurité techniques, suivez les instructions appropriées.

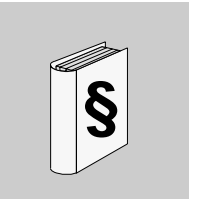
La non-utilisation du logiciel Schneider Electric ou d'un logiciel approuvé avec nos produits peut entraîner des blessures, des dommages ou un fonctionnement incorrect.

Le non-respect de cette consigne peut entraîner des lésions corporelles ou des dommages matériels.

Table des matières



Consignes de sécurité



Informations importantes

AVIS

Lisez attentivement ces instructions et examinez le matériel pour vous familiariser avec l'appareil avant de tenter de l'installer, de le faire fonctionner ou d'assurer sa maintenance. Les messages spéciaux suivants que vous trouverez dans cette documentation ou sur l'appareil ont pour but de vous mettre en garde contre des risques potentiels ou d'attirer votre attention sur des informations qui clarifient ou simplifient une procédure.

L'apposition de ce symbole à un panneau de sécurité Danger ou Avertissement signale un risque électrique pouvant entraîner des lésions corporelles en cas de non-respect des consignes.

Ceci est le symbole d'une alerte de sécurité. Il vous avertit d'un risque de blessures corporelles. Respectez scrupuleusement les consignes de sécurité associées à ce symbole pour éviter de vous blesser ou de mettre votre vie en danger.

⚠ DANGER

DANGER indique une situation immédiatement dangereuse qui, si elle n'est pas évitée, **entraînera** la mort ou des blessures graves.

⚠ AVERTISSEMENT

L'indication **AVERTISSEMENT** signale une situation potentiellement dangereuse et susceptible **d'entraîner la** mort ou des blessures graves.

⚠ ATTENTION

L’indication **ATTENTION** signale une situation potentiellement dangereuse et susceptible **d’entraîner des** blessures d'ampleur mineure à modérée.

ATTENTION

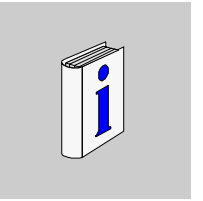
L'indication **ATTENTION**, utilisée sans le symbole d'alerte de sécurité, signale une situation potentiellement dangereuse et susceptible **d'entraîner des** dommages aux équipements.

REMARQUE IMPORTANTE

L’installation, l’utilisation, la réparation et la maintenance des équipements électriques doivent être assurées par du personnel qualifié uniquement. Schneider Electric décline toute responsabilité quant aux conséquences de l’utilisation de cet appareil.

Une personne qualifiée est une personne disposant de compétences et de connaissances dans le domaine de la construction et du fonctionnement des équipements électriques et installations et ayant bénéficié d'une formation de sécurité afin de reconnaître et d’éviter les risques encourus.

A propos de ce manuel



Présentation

Objectif du document

Ce manuel décrit les éléments nécessaires à la programmation des automates Premium, Atrium et Quantum par le biais de l'atelier de programmation Unity Pro.

Champ d'application

Cette documentation est applicable à partir de Unity Pro 5.0.

Information spécifique au produit

⚠ AVERTISSEMENT

FONCTIONNEMENT D'EQUIPEMENT NON INTENTIONNEL

L'utilisation de ce produit requiert une expertise dans la conception et la programmation des systèmes de contrôle. Seules les personnes avec l'expertise adéquate sont autorisées à programmer, installer, modifier et utiliser ce produit.

Respectez les codes et normes de sécurité en vigueur au niveau local et national.

Le non-respect de ces instructions peut provoquer la mort, des blessures graves ou des dommages matériels.

Commentaires utilisateur

Envoyez vos commentaires à l'adresse e-mail techpub@schneider-electric.com

Présentation générale de Unity Pro

I

Présentation

1

Vue d'ensemble

Ce chapitre décrit la structure générale et le comportement général d'un projet créé avec Unity Pro.

Contenu de ce chapitre

Ce chapitre contient les sujets suivants :

Fonctions de Unity Pro

Plateformes matérielles

Unity Pro prend en charge les plateformes matérielles suivantes :
- Modicon M340
- Premium
- Atrium
- Quantum

Langages de programmation

Unity Pro propose les langages suivants pour la création du programme utilisateur :
- Langage à blocs fonction (FBD)
- Langage à contacts (LD)
- Liste d'instructions IL
- Littéral structuré ST
- Diagramme fonctionnel en séquence SFC

Tous ces langages peuvent être utilisés ensemble dans le même projet.

Tous ces langages sont conformes à la norme CEI 61131-3.

Bibliothèques de blocs

Les blocs des nombreuses bibliothèques de blocs comprises dans l'offre de Unity Pro vont des blocs pour opérations booléennes simples aux blocs de commande de boucles de régulation complexes, en passant par des blocs pour chaînes de caractères (strings) et opérations de zones (matrice).

Par souci de clarté, les différents blocs sont structurés en bibliothèques, elles-mêmes structurées en familles.

Les blocs peuvent être utilisés dans les langages FBD, LD, IL et ST.

Eléments d'un programme

Un programme peut se composer :
- d'une tâche maître (MAST) ;
- d'une tâche rapide (FAST) ;
- d'une à quatre tâches auxiliaires (non disponibles pour Modicon M340) ;
- de sections auxquelles est affectée l'une des tâches définies ;
- de sections dédiées au traitement des événements temporisés (Timerx) ;
- de sections de traitement d'événements issus de modules d'entrées/sorties (EVTx) ;
- de sections de sous-programme (SR).

Progiciels

Les progiciels disponibles sont les suivants :

- Unity Pro S
- Unity Pro M
- Unity Pro L
- Unity Pro XL
- Unity Pro XLS
- Unity Developers Edition (UDE)

Evaluation des performances

Le tableau suivant présente les propriétés principales des différents progiciels :

	Unity Pro S	**Unity Pro M**	**Unity Pro L**	**Unity Pro XL**	**Unity Pro XLS**
Langages de programmation					
Langage à blocs fonction (FBD)	+	+	+	+	+
Langage à contacts (LD)	+	+	+	+	+
Liste d'instructions IL	+	+	+	+	+(2)
Littéral structuré ST	+	+	+	+	+(2)
Diagramme fonctionnel en séquence SFC	+	+	+	+	+(2)
Bibliothèques (1)					
Bibliothèque standard	+	+	+	+	+(2)
Bibliothèque de régulation	+	+	+	+	+(2)
Bibliothèque de communication	+	+	+	+	+(2)
Bibliothèque de diagnostics	+	+	+	+	+(2)
Bibliothèque de gestion des E/S	+	+	+	+	+(2)
Bibliothèque système	+	+	+	+	+(2)
Bibliothèque de commande d'entraînement	-	+	+	+	+(2)
Bibliothèque TCP Open	-	En option	En option	En option	En option (2)
Bibliothèque obsolète	+	+	+	+	+(2)
Bibliothèque MFB	+	+	+	+	+(2)
Bibliothèque de sécurité	-	-	-	-	+

	Unity Pro S	**Unity Pro M**	**Unity Pro L**	**Unity Pro XL**	**Unity Pro XLS**
Bibliothèque de gestion des fichiers de carte mémoire	+	+	+	+	+(2)
Informations générales					
Création et utilisation des structures de données (DDT)	+	+	+	+	+(2)
Création et utilisation des blocs fonction dérivés (DFB)	+	+	+	+	+(2)
Navigateur de projet avec vue structurelle et/ou fonctionnelle	+	+	+	+	+
Gestion des droits d'accès	+	+	+	+	+
Ecrans d'exploitation	+	+	+	+	+
Visualisation du diagnostic	+	+	+	+	+
Diagnostic système	+	+	+	+	+
Diagnostic du projet	+	+	+	+	+(2)
Convertisseur d'applications	-	Convertisseur PL7	Convertisseur PL7 Convertisseur Concept	Convertisseur PL7 Convertisseur Concept	Convertisseur PL7 Convertisseur Concept
Gestion de plusieurs stations	-	-	-	-	-
Plateformes prises en charge					
Modicon M340	BMX P34 1000 BMX P34 20••	BMX P34 1000 BMX P34 20••	BMX P34 1000 BMX P34 20••	BMX P34 1000 BMX P34 20••	BMX P34 1000 BMX P34 20••
Premium	-	P57 0244M P57 CA 0244M P57 CD 0244M P57 104M P57 154M P57 1634M P57 204M P57 254M P57 2634M H57 24M	Toutes les UC **sauf** : P57 554M P57 5634M	Toutes les UC	Toutes les UC

	Unity Pro S	Unity Pro M	Unity Pro L	Unity Pro XL	Unity Pro XLS
Quantum	-	-	140 CPU 311 10 140 CPU 434 12 U/A* 140 CPU 534 14 U/A* * Mise à niveau à l'aide du système d'exploitation Unity	CPU 311 10 CPU 534 14 U/A CPU 651 50 CPU 652 60 CPU 651 60 CPU 671 60	CPU 311 10 CPU 434 12 U/A CPU 534 14 U/A CPU 651 50 CPU 651 60 CPU 652 60 CPU 671 60 CPU 651 60 S CPU 671 60 S CPU 672 61
Atrium	-	PCI 57 204	Toutes les UC	Toutes les UC	Toutes les UC
Simulateur	+	+	+	+	+
Transparence					
Liens hypertexte	+	+	+	+	+
Serveur Unity Pro (pour OFS, UDE, UAG)	-	-	-	+	+
Composants logiciels inclus dans le progiciel					
Documentation sous forme d'aide contextuelle et en version PDF	+	+	+	+	+
Outil OS Loader + Micrologiciel matériel	+	+	+	+	+
Unity Loader	+	+	+	+	+

+ = disponible
+ (1) = la disponibilité des blocs dépend des plateformes matérielles.
+ (2) = Disponible sur tous les automates à l'exception des plateformes CPU 651 60 S et CPU 671 60 S.
- = non disponible

Convention de nommage

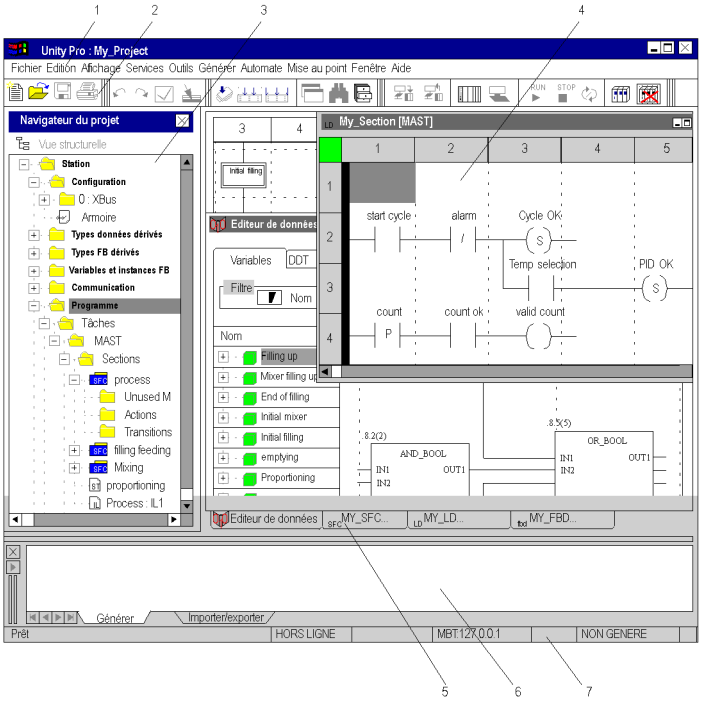
Dans la documentation suivante, "Unity Pro" est un terme générique pour "Unity Pro S", "Unity Pro M", "Unity Pro L", "Unity Pro XL" et "Unity Pro XLS".

Interface utilisateur

Présentation

L'interface utilisateur se compose de plusieurs fenêtres et barres d'outils pouvant être positionnées librement.

Interface utilisateur :



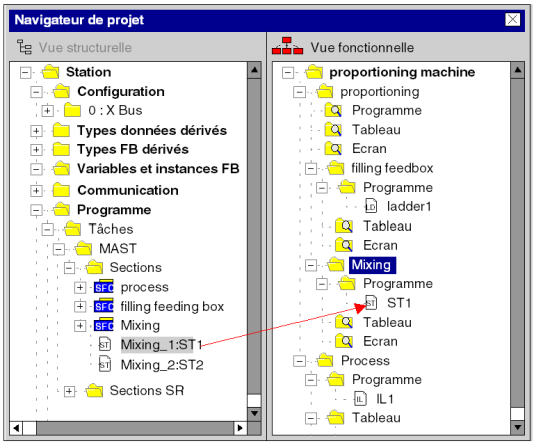
Légende :

Numéro	Description
1	Barre de menus *(voir Unity Pro, Modes de marche,)*
2	Barre d'outils *(voir Unity Pro, Modes de marche,)*
3	Navigateur du projet *(voir Unity Pro, Modes de marche,)*
4	Fenêtre de l'éditeur (éditeurs de langages, éditeur de données, etc.)
5	Onglets d'accès direct aux fenêtres de l'éditeur
6	Fenêtre d'information *(voir Unity Pro, Modes de marche,)* (donne des informations sur les erreurs survenues, le suivi des signaux, les fonctions d'importation, etc.)
7	Ligne d'état *(voir Unity Pro, Modes de marche,)*

Navigateur de projet

Introduction

Le navigateur de projet affiche tous les paramètres du projet. L'affichage peut se présenter sous forme structurelle (topologique) et/ou fonctionnelle.



Vue structurelle

Dans l'affichage structurel, le navigateur de projet propose entre autres les fonctions suivantes :

- création et suppression d'éléments
- Le symbole de section affiche le langage de programmation de section et sa programmation éventuelle (dans le cas d'une section vide, le symbole est grisé)
- affichage des propriétés des éléments
- création de répertoires utilisateur
- démarrage des différents éditeurs
- démarrage de la fonction import/export

Vue fonctionnelle

Dans l'affichage fonctionnel, le navigateur de projet propose entre autres les fonctions suivantes :

- création de modules fonctionnels
- insertion de sections, tables d'animation, etc. par glisser-lâcher à partir de l'affichage structurel
- création de sections
- affichage des propriétés des éléments
- démarrage des différents éditeurs
- symbole de section indiquant le langage de programmation et d'autres attributs.

Formats de fichier d'application utilisateur et de projet

Introduction

Unity Pro gère trois types de fichiers pour le stockage des applications utilisateur et des projets. Chaque type de fichier peut être utilisé en fonction de conditions précises.

Les types de fichiers sont reconnaissables à leur extension :

- **.STU* : fichier Unity Pro
- **.STA* : fichier Unity Pro Archived Application
- **.XEF* : fichier Unity Pro Application Exchange

Fichier STU

Ce type de fichier est utilisé pour les tâches quotidiennes. Ce format est utilisé par défaut à l'ouverture ou à l'enregistrement d'un projet utilisateur.

Le tableau suivant présente les avantages et les inconvénients du fichier *STU* :

Avantages	Inconvénients
• Le projet peut être enregistré à toute étape (cohérente ou non) à l'aide de la commande par défaut.	• Peu pratique lors du transfert du projet, en raison de la taille importante du fichier.
• L'enregistrement et l'ouverture des projets est rapide et toute la base de données interne est présente dans le fichier.	• Non compatible lors de la mise à niveau de Unity Pro.

Fichier STA

Ce type de fichier est utilisé pour archiver les projets et ne peut être créé qu'après la génération du projet. Ce type de fichier est compatible avec les versions ultérieures de Unity Pro.

Il existe deux manières de créer un fichier **STA** :

- Pour créer un fichier **STA manuellement**, choisissez le menu **Fichier** → **Archiver** dans la fenêtre principale de Unity Pro.
- Un fichier **STA** est créé **automatiquement** chaque fois que le projet est enregistré sous forme de fichier **STU** dans son état **généré**.

NOTE : le fichier STA créé automatiquement est enregistré dans le même répertoire et avec le même nom de fichier que le fichier de projet STU, à l'exception du **suffixe « .Auto »** ajouté au nom de fichier. Si un fichier STA automatique existe déjà, **il est remplacé** sans qu'une confirmation soit demandée.

NOTE : si le projet est à l'état **généré**, l'enregistrement d'un fichier STU par l'intermédiaire d'un serveur Unity Pro crée également un fichier STA.

Pour ouvrir un fichier *STA*, choisissez le menu **Fichier** →**Ouvrir** dans la fenêtre principale de Unity Pro.

NOTE : dans la fenêtre du menu **Ouvrir**, le type de fichier sélectionné doit être *Fichier d'archive (*.STA)*.

- Pour plus d'informations sur la création d'un fichier *STA*, consultez le Manuel d'installation Unity Pro : Créer un fichier Unity Pro Archived Application.
- Pour plus d'informations sur l'ouverture d'un fichier *STA*, consultez le Manuel d'installation Unity Pro : Restauration de fichiers Unity Pro Archived Application.

Le tableau suivant présente les avantages et les inconvénients du fichier *STA* :

Avantages	**Inconvénients**
• Enregistrement rapide du projet.	• Ne peut être créé qu'après la génération du projet.
• Les projets peuvent être partagés par courrier électronique ou à l'aide de supports mémoire de petite taille.	• L'ouverture du projet est longue, car le fichier de projet est régénéré avant le fonctionnement.
• Possibilité de se connecter à l'automate en mode connecté égal après l'ouverture du projet sur une nouvelle version de Unity Pro. Pour plus d'informations, consultez la section Connexion/Déconnexion *(voir Unity Pro, Modes de marche,)* dans le manuel Modes de marche *(voir Unity Pro, Modes de marche,)*.	
• Permettre les modifications en ligne avec l'automate sans téléchargement préalable sur l'automate.	
• Un fichier *STA* généré est compatible avec toutes les versions de Unity Pro. **NOTE :** pour charger un fichier *STA* créé avec une autre version de Unity Pro, toutes les fonctions utilisées dans l'application doivent être prises en charge dans la version courante.	

Fichier XEF

Ce type de fichier est utilisé pour exporter des projets dans un format source *XML* et peut être créé à toutes les étapes du projet.

Pour exporter un fichier *XEF*, choisissez le menu **Fichier** →**Exporter le projet** dans la fenêtre principale de Unity Pro.

Pour importer un fichier *XEF*, choisissez le menu **Fichier** →**Ouvrir** dans la fenêtre principale de Unity Pro.

NOTE : dans la fenêtre du menu **Ouvrir**, le type de fichier sélectionné doit être *Fichier d'échange d'application (*.XEF)*.

Pour plus d'informations sur la création d'un fichier *XEF*, consultez le Manuel d'installation Unity Pro : Créer un fichier Unity Pro Application Exchange.

Pour plus d'informations sur la restauration d'un fichier *XEF*, consultez le Manuel d'installation Unity Pro : Restauration d'un fichier Unity Pro Application Exchange.

Le tableau suivant présente les avantages et les inconvénients du fichier *XEF* :

Avantages	Inconvénients
• Le format source *XML* garantit la compatibilité du projet avec toutes les versions de Unity Pro.	• Taille moyenne.
	• L'ouverture du projet prend du temps car le projet est importé avant de fonctionner.
	• La génération du projet est obligatoire pour réassembler le code binaire du projet.
	• Le fonctionnement avec l'automate nécessite la recompilation de tout le projet et son téléchargement sur le processeur.
	• Il est impossible de connecter l'automate en mode connecté égal avec un fichier *XEF*. Pour plus d'informations, consultez la section Connexion/Déconnexion *(voir Unity Pro, Modes de marche,)* dans le manuel Modes de marche *(voir Unity Pro, Modes de marche,)*.

Informations importantes

Les fichiers *STU* ne sont pas compatibles d'une version de Unity Pro à l'autre. Afin d'utiliser un projet avec d'autres versions de Unity Pro, les utilisateurs doivent créer des :

- fichiers Unity Pro Archived Application *(STA)* :
  Avec le fichier *STA*, il est possible de réutiliser le projet généré courant avec la nouvelle version de Unity Pro installée sur l'ordinateur.
- fichiers d'échange d'application Unity Pro *(XEF)* :
  Les fichiers *XEF* doivent être utilisés si le projet a été généré.

Comparaison des types de fichiers

Le tableau suivant récapitule les trois types de fichiers :

Types de fichiers	*STU*	*STA*	*XEF*
Applications binaires	Oui	Oui	Non
Applications sources	Oui	Oui	Oui
Base de données interne	Oui	Non	Non
Taille de fichier	10, voir (1)	0,03, voir (1)	3
Temps d'enregistrement (comparatif)	10	1.6	6
Temps d'ouverture (comparatif)	1	10	10
Connexion à l'automate en mode connecté égal	Possible	Possible	Impossible, voir (2)
Sauvegarde du fichier	Possible	Possible, voir (3)	Possible

(1) : Fichiers compressés.

(2) : Le projet doit d'abord être chargé sur l'automate.

(3) : Le projet ne peut être enregistré que s'il a été généré.

NOTE : les valeurs de la table représentent le rapport entre les types de fichiers, la valeur du *STU* tenant lieu de référence.

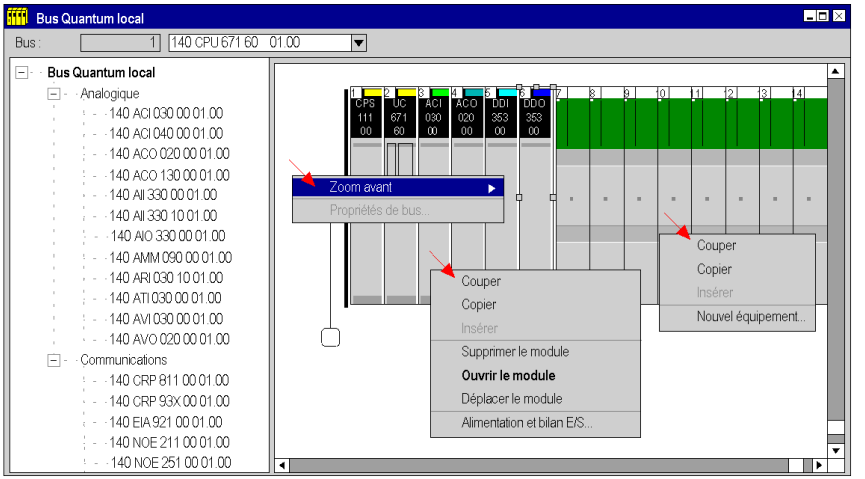
Configurateur

Fenêtre du configurateur

La fenêtre du configurateur est divisée en deux parties :

- Fenêtre catalogue
  Un module peut être sélectionné dans cette fenêtre et être inséré directement par glisser-lâcher dans la représentation graphique de la configuration de l'automate.
- Représentation graphique de la configuration de l'automate

Représentation de la fenêtre du configurateur



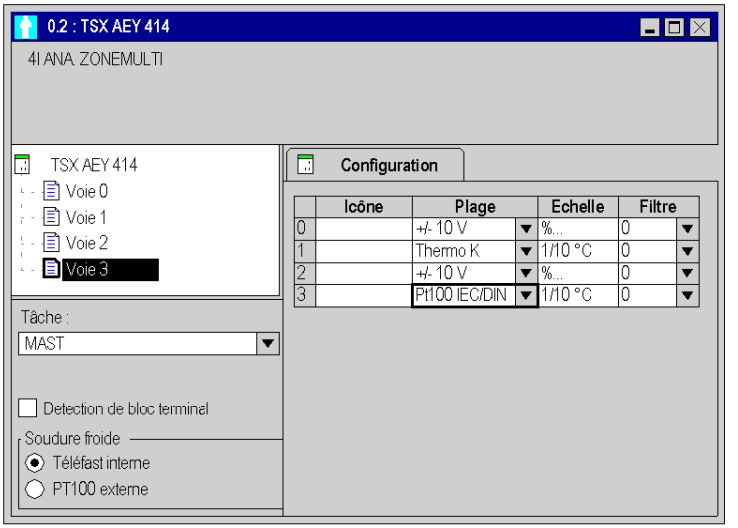
Selon la position du pointeur de la souris, l'un des menus contextuels suivants est appelé :

- si le pointeur de la souris est sur l'arrière-plan, il permet entre autres :
  - de modifier l'UC
  - de sélectionner divers facteurs de zoom
- si le pointeur de la souris est sur un module, il permet entre autres :
  - d'accéder aux fonctions d'édition (supprimer, copier, déplacer)
  - d'ouvrir la configuration d'un module afin de définir les paramètres spécifiques à ce module
  - d'afficher les propriétés E/S et le bilan électrique
- si le pointeur de la souris est sur un emplacement vide, il permet entre autres :
  - d'insérer un module du catalogue
  - d'insérer un module préalablement copié, avec ses propriétés définies

Configuration module

La fenêtre de configuration module (appel via le menu contextuel du module ou double-clic sur le module) permet de configurer le module. Font par exemple partie de la configuration la sélection de la voie, la sélection de la fonction de la voie choisie, l'affectation d'adresses de mémoire State Ram (quantum seulement), etc.

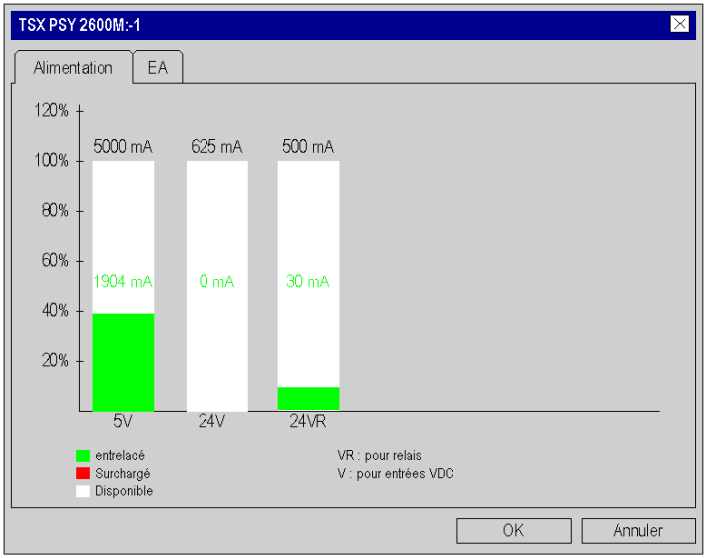
Fenêtre de configuration d'un module E/S Premium :



Propriétés de module

La fenêtre des propriétés de module (appel par le menu contextuel du module) indique les propriétés du module, comme la consommation électrique, le nombre de points E/S (Premium seulement), etc.

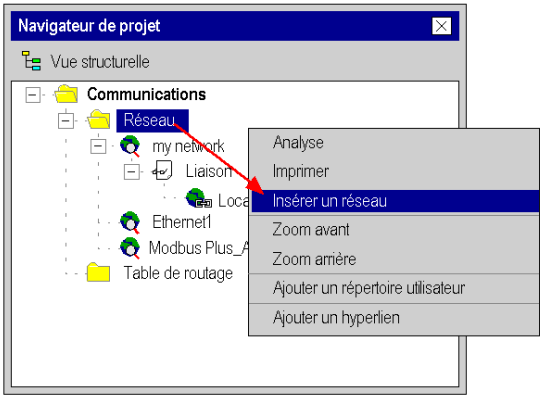
La fenêtre des propriétés de module dédiée à l'alimentation indique le bilan électrique du rack :



Configuration réseau

La configuration réseau s'appelle via le dossier communication.

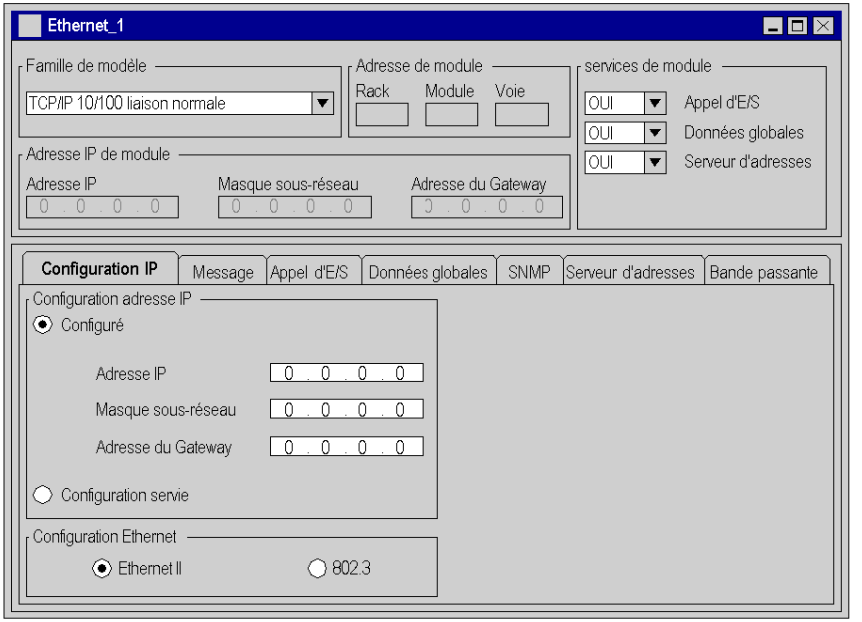
Configuration réseau :



Le menu contextuel de la configuration réseau permet entre autres les fonctions suivantes :

- création de réseaux
- analyse du réseau
- impression de la configuration réseau

Fenêtre de configuration d'un réseau :



A l'issue de la configuration, le réseau est affecté à un module de communication.

Editeur de données

Introduction

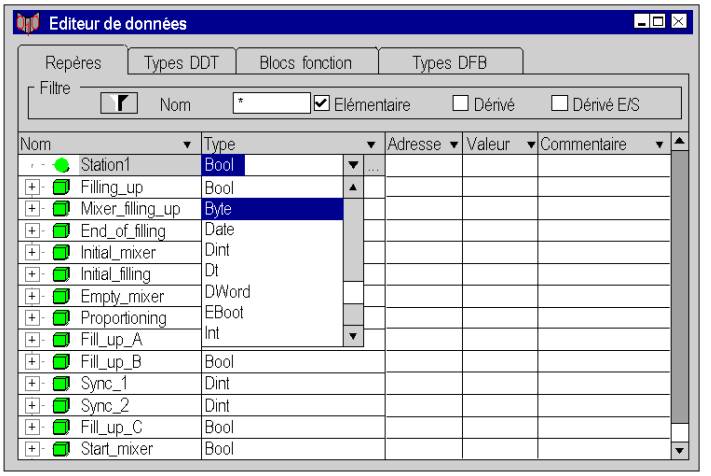
L'éditeur de données propose les fonctions suivantes :
- déclaration d'instances de variable,
- définition de types de données dérivés (DDT),
- déclaration d'instance de blocs fonction élémentaires et dérivés (EFB/DFB),
- définition des paramètres de blocs fonction dérivés (DFB).

Les fonctions suivantes sont disponibles dans tous les onglets de l'éditeur de données :
- copier, couper, coller,
- étendre/assembler des données structurées,
- trier en fonction du type, du symbole, de l'adresse, etc.,
- filtrer,
- insérer, supprimer et modifier la position de colonnes,
- glisser-lâcher entre l'éditeur de données et les éditeurs de programme,
- annuler la dernière modification,
- importer/exporter.

Variables

L'onglet **Variables** permet de déclarer les variables.

Onglet Variables :

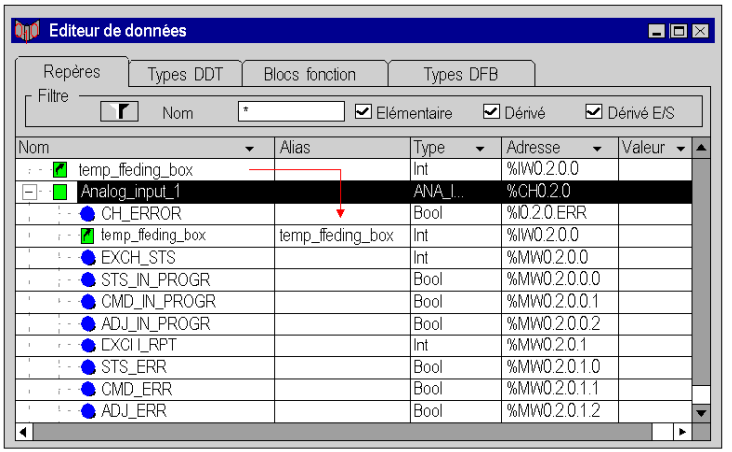


Les fonctions disponibles sont les suivantes :
- définition d'un symbole pour les variables,
- affectation d'un type de données,
- boîte de dialogue de sélection personnalisée pour les types de données dérivés,
- affectation d'une adresse,
- symbolisation automatique des variables E/S,
- affectation d'une valeur initiale,
- saisie d'un commentaire,
- affichage de toutes les propriétés d'une variable dans une boîte de dialogue distincte dédiée aux propriétés.

Types de données dépendant du matériel (IODDT)

Les IODDT servent à affecter la structure E/S complète d'un module à une variable unique.

Affectation des IODDT :



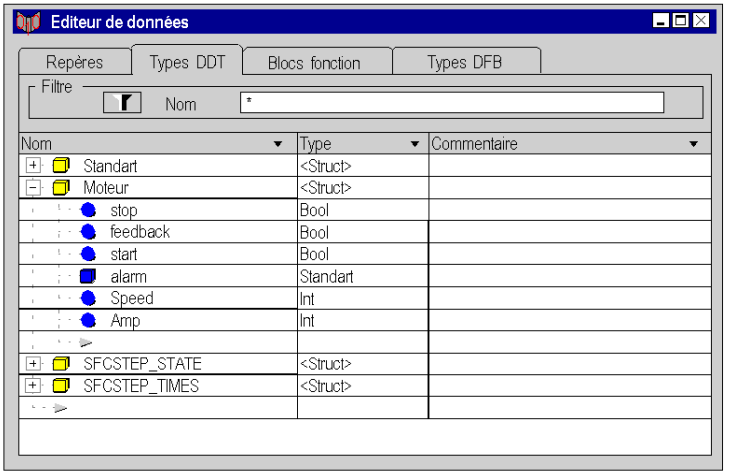
Les fonctions disponibles sont les suivantes :
- Les IO DDT permettent d'affecter des structures E/S complètes à une variable unique.
- Après la saisie de l'adresse des variables, le mot ou bit d'entrée/sortie est automatiquement affecté à tous les éléments de la structure.
- La possibilité d'affecter ultérieurement des adresses permet de créer facilement des modules standard dont les adresses ne sont définies que plus tard.
- Un nom d'alias (pseudonyme) peut être attribué à tous les éléments d'une structure IODDT.

Types de données dérivés (DDT)

L'onglet **Types DDT** permet de définir les types de données dérivés (DDT).

Un type de données dérivé est la définition d'une structure ou d'une zone de tout type de données déjà défini (élémentaire ou dérivé).

Onglet **DDT types** :



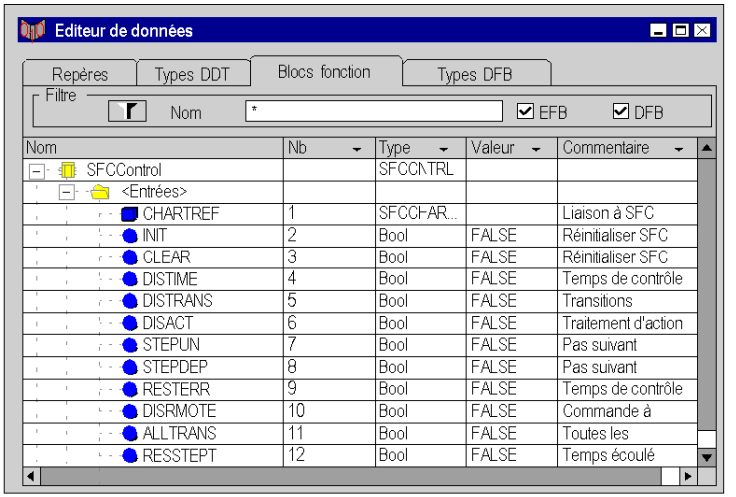
Les fonctions disponibles sont les suivantes :

- définition de DDT imbriqués (max. 8 niveaux),
- définition de tableaux ayant jusqu'à six dimensions,
- affectation d'une valeur initiale,
- affectation d'une adresse,
- saisie d'un commentaire,
- analyse du type de données dérivé,
- affectation du type de données dérivé à une bibliothèque,
- affichage de toutes les propriétés d'un type de données dérivé dans une boîte de dialogue distincte dédiée aux propriétés.
- Un nom d'alias (pseudonyme) peut être attribué à tous les éléments d'un tableau ou d'une structure DDT.

Blocs fonction

L'onglet **Blocs fonction** est utilisé pour la déclaration d'instance de blocs fonction élémentaires et dérivés (EFB/DFB).

Onglet **Blocs fonction** :



Nom	Nb	Type	Valeur	Commentaire
SFCControl		SFCCNTRL		
<Entrées>				
CHARTREF	1	SFCCHAR...		Liaison à SFC
INIT	2	Bool	FALSE	Réinitialiser SFC
CLEAR	3	Bool	FALSE	Réinitialiser SFC
DISTIME	4	Bool	FALSE	Temps de contrôle
DISTRANS	5	Bool	FALSE	Transitions
DISACT	6	Bool	FALSE	Traitement d'action
STEPUN	7	Bool	FALSE	Pas suivant
STEPDEP	8	Bool	FALSE	Pas suivant
RESTERR	9	Bool	FALSE	Temps de contrôle
DISRMOTE	10	Bool	FALSE	Commande à
ALLTRANS	11	Bool	FALSE	Toutes les
RESSTEPT	12	Bool	FALSE	Temps écoulé

Les fonctions disponibles sont les suivantes :

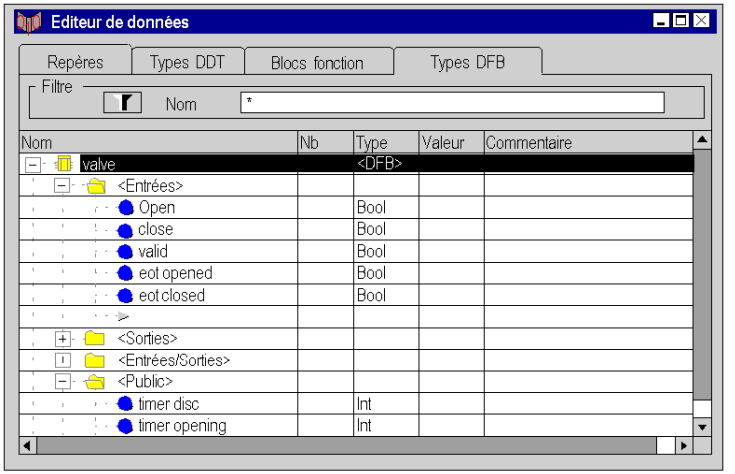
- affichage des blocs fonction utilisés dans le projet,
- définition d'un symbole pour les blocs fonction utilisés dans le projet,
- copie automatique des symboles définis dans le projet,
- saisie d'un commentaire relatif aux blocs fonction,
- affichage de tous les paramètres (entrées/sorties) des blocs fonction,
- affectation d'une valeur initiale aux entrées/sorties de bloc fonction.

Types DFB

L'onglet **Types DFB** est utilisé pour définir les paramètres des blocs fonction dérivés (DFB).

La création de la logique DFB s'effectue directement dans une ou plusieurs sections des langages de programmation FBD, LD ou ST.

Onglet **Types DFB** :



Les fonctions disponibles sont les suivantes :

- définition du nom DFB,
- définition de tous les paramètres du DFB, tels que les :
  - entrées,
  - sorties,
  - VAR_IN_OUT (entrées/sorties combinées),
  - variables privées,
  - variables publiques,
- affectation du type de données aux paramètres DFB,
- boîte de dialogue de sélection personnalisée pour les types de données dérivés,
- affectation d'une valeur initiale,
- imbrication de DFB,
- utilisation de plusieurs sections dans un DFB,
- saisie d'un commentaire relatif aux DFB et paramètres DFB,
- analyse des DFB définis,
- gestion de versions,
- affectation des DFB définis à une bibliothèque.

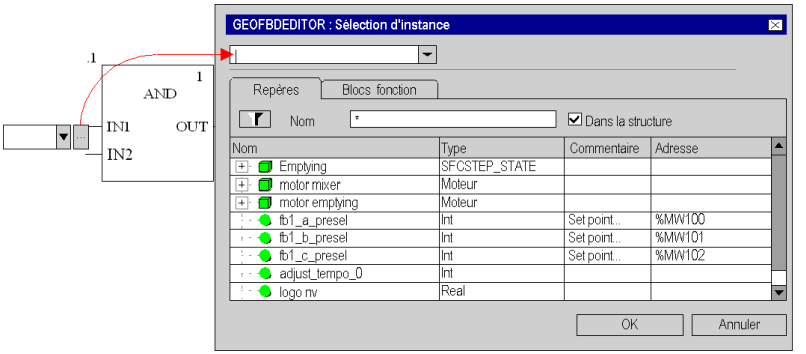
Utilisation des données

Les instances et types de données créés avec l'éditeur de données peuvent être insérés dans les éditeurs de langage de programmation en fonction du contexte.

Les fonctions disponibles sont les suivantes :

- accès à tous les éditeurs de langage de programmation,
- affichage uniquement des données compatibles,
- affichage des fonctions, blocs fonction, procédures et types de données dérivés classés en fonction de leur appartenance aux bibliothèques,
- déclaration d'instance possible pendant la programmation.

Boîte de dialogue de sélection de données :



Modifications en ligne

Il est possible de modifier le type d'une variable ou d'une instance de bloc fonction (FB) déclarée dans l'application ou dans un bloc fonction dérivé (DFB) directement en mode connecté. Cela signifie qu'il n'est pas nécessaire d'arrêter l'application pour effectuer ce type de modification.

Ces opérations peuvent être réalisées dans l'éditeur de données ou dans l'éditeur de propriétés comme en mode déconnecté.

⚠ ATTENTION
COMPORTEMENT INATTENDU DE L'APPLICATION Lors du changement du type d'une variable, la nouvelle valeur de la variable à modifier dépend de son type : ● Dans le cas d'une **variable non affectée**, la variable prend sa valeur initiale, si elle existe. Sinon, elle prend la valeur par défaut. ● Dans le cas d'une **variable affectée**, la variable redémarre avec la valeur initiale, si elle existe. Sinon, elle conserve la valeur binaire courante. Avant d'appliquer le changement de type de variable, vérifiez l'effet de la nouvelle valeur sur l'exécution de l'application. **Le non-respect de ces instructions peut provoquer des blessures ou des dommages matériels.**

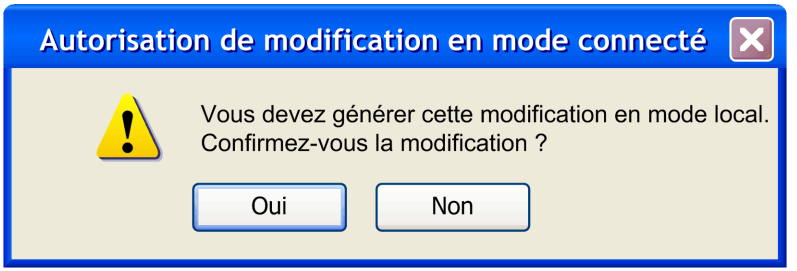
NOTE : il est impossible de modifier le type d'une variable déclarée en type de données dérivé (DDT) en mode connecté. L'application doit être mise en mode local pour effectuer cette modification.

Restrictions sur les modifications en mode connecté

Dans les cas suivants, la modification en mode connecté d'une variable ou d'un bloc fonction (FB) n'est pas autorisée :

- Si la variable est utilisée comme données globales de réseau, la modification du type en mode connecté n'est pas autorisée.
- Si le FB courant ne peut pas être retiré en mode connecté, ou si un nouveau FB ne peut pas être ajouté en mode connecté, la modification du type de ce FB en mode connecté n'est pas autorisée. En effet, certains blocs fonction élémentaires (EFB) tels que les blocs fonction standard (SFB) ne peuvent pas être ajoutés ou retirés en mode connecté. Par conséquent, la modification d'une instance d'EFB en une instance de SFB est impossible, et réciproquement.

Dans ces deux cas, la boîte de dialogue suivante apparaît :



NOTE : en raison de ces limites, si un bloc fonction dérivé (DFB) contient au moins une instance d'un SFB, il n'est pas possible d'ajouter ou de retirer une instance de ce DFB en mode connecté.

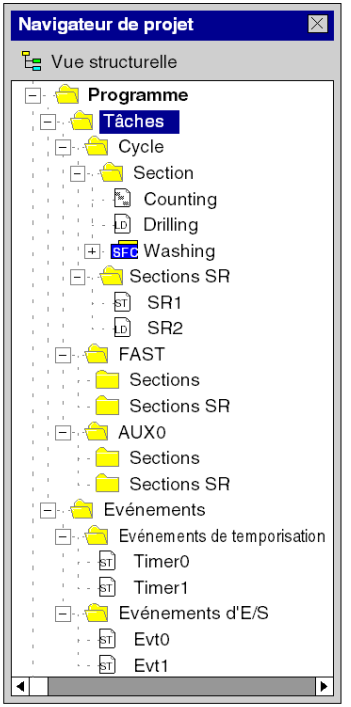
Editeur

Introduction

Un programme peut se composer de :

- **tâches** exécutées de manière cyclique ou périodique.
  Les tâches sont elles-mêmes composées de :
  - sections
  - sous-programmes
- **traitements événementiels** exécutés prioritairement à toutes les autres tâches.
  Les traitements événementiels sont eux-mêmes composés de :
  - sections de traitement d'événements à commande temporelle
  - sections de traitement d'événements à commande matérielle.

Exemple de programme :



Tâches

Unity Pro gère des tâches multiples (multitâche).

Les tâches sont exécutées " en parallèle " indépendamment les unes des autres, avec les niveaux de priorité d'exécution commandés par l'API. Les tâches peuvent être adaptées à des exigences diverses et constituent ainsi un instrument puissant de structuration du projet.

Un projet multitâche peut se composer :
- d'une tâche maître (MAST)
  L'exécution de la tâche maître est cyclique ou périodique.
  Constituant la partie principale du programme, elle est exécutée de manière séquentielle.
- d'une tâche rapide (FAST)
  L'exécution de la tâche rapide est périodique. Son niveau de priorité est plus élevé que celui de la tâche maître. La tâche rapide est destinée aux traitements de courte durée et périodiques.
- 1 à 4 tâches auxiliaires (AUX)
  L'exécution des tâches auxiliaires est périodique. Elles sont destinées aux traitements plus lents, ce sont les tâches les moins prioritaires.

Le projet peut aussi se composer d'une seule tâche. Dans ce cas, seule la tâche maître est active.

Traitement événementiel

Le traitement événementiel s'effectue dans des sections dites d'événement. Ces sections d'événement sont exécutées en priorité sur les sections de toutes les autres tâches. Elles conviennent donc aux traitements demandant des délais de réaction très courts par rapport à l'arrivée de l'événement.

Les types de section disponibles pour le traitement événementiel sont les suivants :
- section de traitement d'événements à commande temporelle (section Timerx)
- section de traitement d'événements à commande matérielle (section Evtx)

Les langages de programmation suivants sont pris en charge :
- FBD (langage en blocs fonctionnels)
- LD (langage à contacts)
- IL (liste d’instructions)
- ST (littéral structuré)

sections

Les sections sont des unités de programme autonomes dans lesquelles est créée la logique du projet.

Les sections sont exécutées dans leur ordre de représentation dans le navigateur de projet (vue structurelle). Elles sont reliées à une tâche.

Une même section ne peut pas appartenir à plusieurs tâches en même temps.

Les langages de programmation suivants sont pris en charge :
- FBD (langage en blocs fonctionnels)
- LD (langage à contacts)
- SFC (diagramme fonctionnel en séquence)
- IL (liste d'instructions)
- ST (littéral structuré)

Sous-programmes

Les sous-programmes sont créés en tant qu'unités distinctes dans des sections de sous-programme.

Les appels aux sous-programmes s'effectuent à partir des sections ou d'un autre sous-programme.

Une imbrication de huit niveaux maximum est possible.

Un sous-programme ne peut pas s'appeler lui-même (non récursif).

Les sous-programmes sont affectés à une tâche. Un même sous-programme ne peut pas être appelé par différentes tâches.

Les langages de programmation suivants sont pris en charge :
- FBD (langage en blocs fonctionnels)
- LD (langage à contacts)
- IL (liste d'instructions)
- ST (littéral structuré)

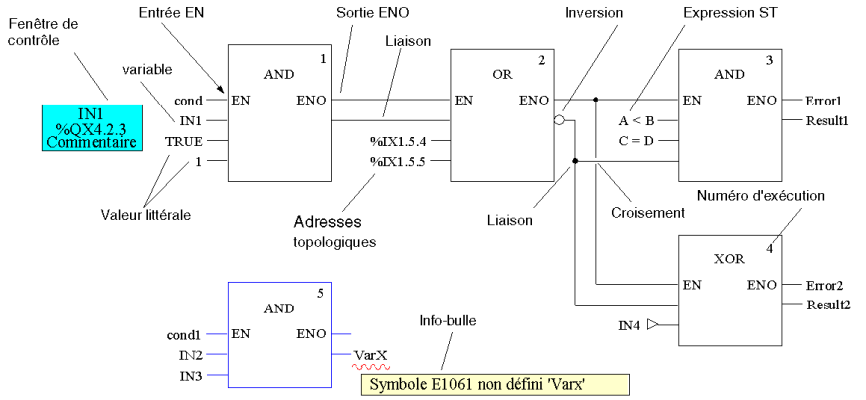
Langage à blocs fonction (FBD)

Introduction

L'éditeur FBD permet la programmation graphique de blocs fonction conformément à la norme CEI 61131-3.

Représentation

Représentation d'une section FBD :



Objets

Les objets du langage de programmation FBD (langage à blocs fonction) offrent des aides permettant de structurer une section en un ensemble de :

- Fonctions élémentaires (EF)
- Blocs fonction élémentaires (EFB)
- Blocs fonction dérivés (DFB)
- Procédures
- Appels de sous-programme
- Sauts
- Liens
- Paramètres réels
- Objets texte pour commenter la logique

Propriétés

Les sections FBD comportent toujours une grille de fond. Une unité de grille comprend 10 points de trame. Une unité de trame est l'espace le plus petit possible entre deux objets d'une section FBD.

Le langage FBD n'est pas basé sur les cellules. Les objets sont toutefois ajustés sur les points de trame.

Une section FBD peut être configurée en nombre de cellules (points de trame horizontaux et points de trame verticaux).

Le programme peut être saisi à l'aide de la souris ou du clavier.

Facilités de saisie

L'éditeur FBD propose les facilités de saisie suivantes :

- barres d'outils permettant un accès rapide et facile aux objets souhaités,
- vérification syntaxique et sémantique pendant l'écriture du programme,
  - affichage en bleu des fonctions et blocs fonction incorrects,
  - soulignement par une ligne rouge ondulée des mots inconnus (ex : variables non déclarées) ou types de données inadaptées,
  - description rapide des erreurs dans une info-bulle,
- les informations sur les variables et les broches peuvent être affichées dans une info-bulle,
  - type, nom, adresse et commentaire d'une variable/expression,
  - type, nom et commentaire d'une broche FFB,
- affichage en tableau des FFB,
- saisie et affichage des paramètres réels sous forme de symboles ou d'adresses topologiques,
- facteurs de zoom différents,
- suivi de liens,
- optimisation des chemins de liaison,
- affichage des fenêtres de contrôle.

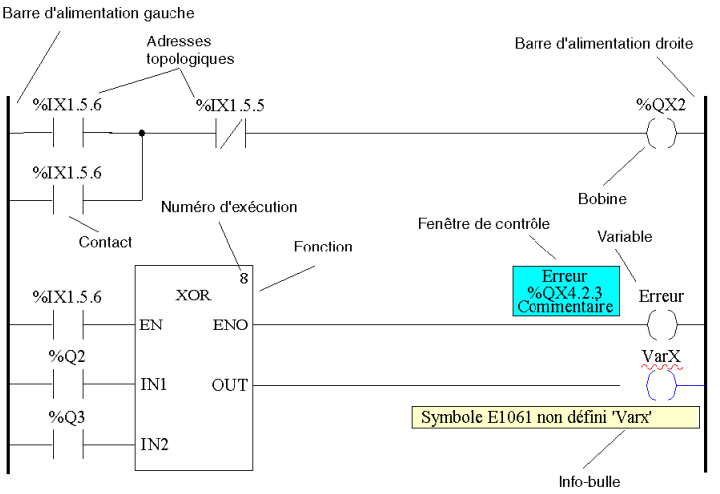
Schéma à contacts (LD)

Introduction

L'éditeur LD permet la programmation graphique de schémas à contacts conformément à la norme CEI 61131-3.

Représentation

Représentation d'une section LD :



Objets

Les objets du langage de programmation LD (Ladder Diagram, schéma à contacts) offrent des aides permettant de structurer une section en un ensemble de :

- contacts,
- bobines,
- fonctions élémentaires (EF),
- blocs fonction élémentaires (EFB),
- blocs fonction dérivés (DFB),
- procédures,
- éléments de commande,
- blocs opération et comparaison constituant une extension de la norme CEI 61131-3,
- appels de sous-programme,
- sauts,
- liens,
- paramètres réels,
- objets texte pour commenter la logique.

Propriétés

Les sections LD disposent d'une grille d'arrière-plan qui divise la section en lignes et en colonnes.

Le langage de programmation LD est axé sur les cellules, ce qui signifie qu'un seul objet peut être placé dans chaque cellule.

Les sections LD peuvent contenir de 11 à 64 colonnes et de 17 à 2 000 lignes.

Le programme peut être saisi à l'aide de la souris ou du clavier.

Facilités de saisie

L'éditeur LD propose les facilités de saisie suivantes :

- les objets peuvent être sélectionnés dans la barre d'outils, le menu ou directement à l'aide de raccourcis clavier,
- vérification syntaxique et sémantique pendant l'écriture du programme,
  - affichage en bleu des objets incorrects,
  - soulignement par une ligne rouge ondulée des mots inconnus (ex : variables non déclarées) ou types de données inadaptées,
  - description rapide des erreurs dans une info-bulle,
- les informations sur les variables et les éléments d'une section LD susceptibles d'être connectés à une variable (broches, contacts, bobines, opération et blocs de comparaison) peuvent être affichées dans une info-bulle,
  - type, nom, adresse et commentaire d'une variable/expression,
  - type, nom et commentaire des broches FFB, des contacts etc.,
- affichage en tableau des FFB,
- saisie et affichage des paramètres réels sous forme de symboles ou d'adresses topologiques,
- facteurs de zoom différents,
- suivi de liens FFB,
- optimisation des chemins de liaison des liens FFB,
- affichage des fenêtres de contrôle.

Présentation générale du langage séquentiel SFC

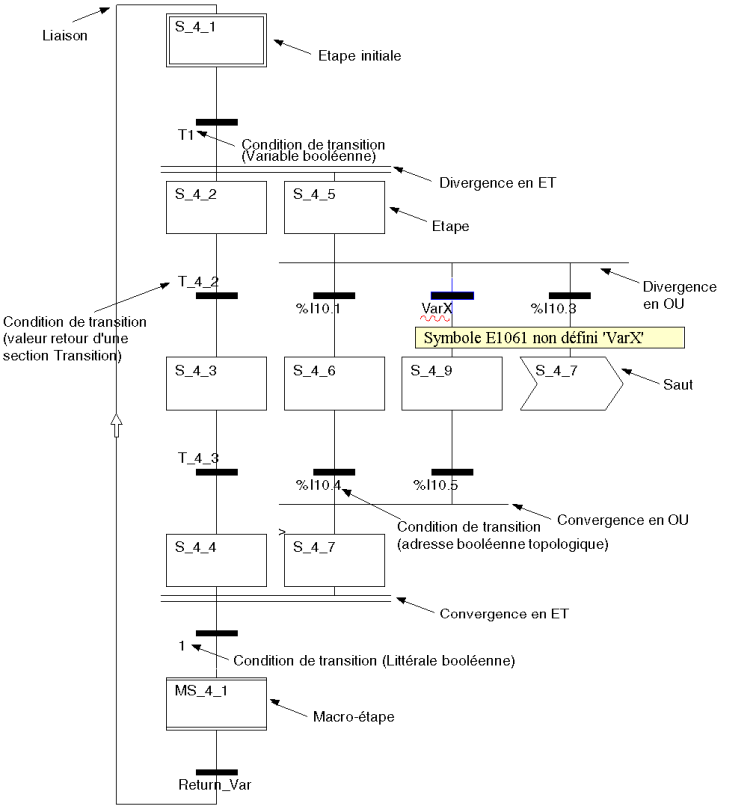
Introduction

Cette section décrit le langage séquentiel SFC (Diagramme fonctionnel en séquence), conforme à la norme CEI 61131-3.

Les restrictions relatives à la conformité CEI peuvent être levées grâce à des procédures d'activation explicites. Des fonctionnalités telles que Multijeton, étapes initiales multiples, saut vers et depuis des chaînes parallèles, etc. sont alors possibles.

Représentation

Représentation d'une section SFC :



Objets

Une section SFC propose les objets suivants pour la création d'un programme :
- étapes,
- macroétapes (séquences de sous-étape intégrées),
- transitions (conditions de transition),
- sections transition,
- sections Action,
- sauts,
- liens,
- séquences alternatives,
- séquences en parallèle,
- objets texte pour commenter la logique.

Propriétés

L'éditeur SFC dispose d'une grille d'arrière-plan qui divise la section en 200 lignes et 32 colonnes.

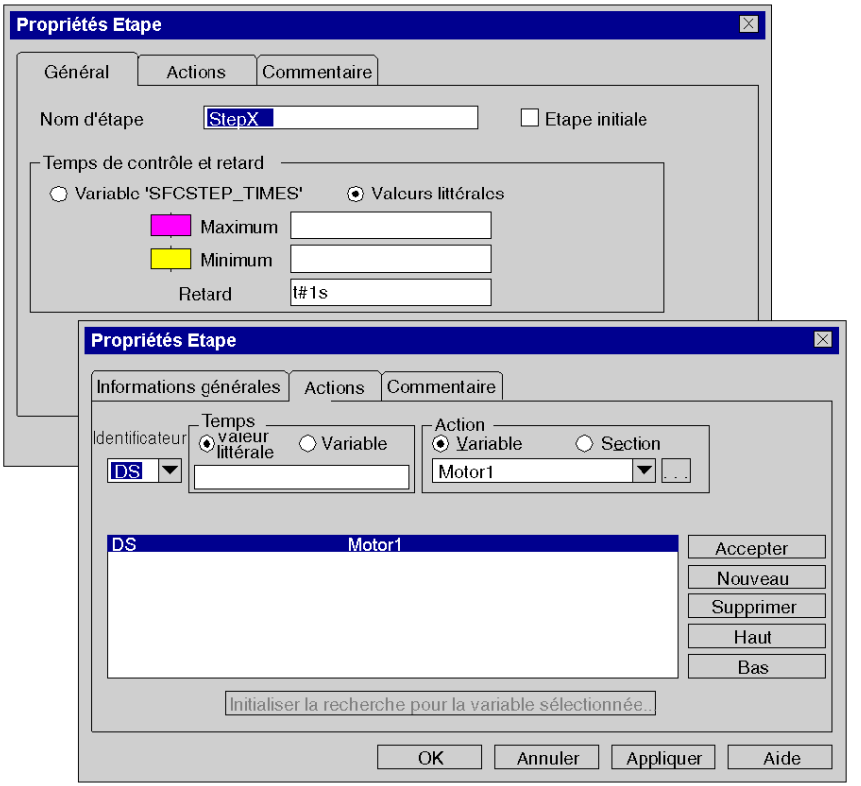
Le programme peut être saisi à l'aide de la souris ou du clavier.

Facilités de saisie

L'éditeur SFC propose les facilités de saisie suivantes :
- barres d'outils permettant un accès rapide et facile aux objets souhaités,
- numérotation automatique des étapes,
- accès direct aux actions et aux conditions de transition,
- vérification syntaxique et sémantique pendant l'écriture du programme,
  - affichage en bleu des objets incorrects,
  - soulignement par une ligne rouge ondulée des mots inconnus (ex : variables non déclarées) ou types de données inadaptées,
  - description rapide des erreurs dans une info-bulle,
- les informations sur les variables et les transitions peuvent être affichées dans une info-bulle,
  - type, nom, adresse et commentaire d'une variable/expression,
  - type, nom et commentaire des transitions,
- facteurs de zoom différents,
- affichage/masquage des actions affectées,
- suivi de liens,
- optimisation des chemins de liaison.

Propriétés Etape

Propriétés Etape :



Les propriétés de l'étape se définissent à l'aide d'une boîte de dialogue proposant les fonctionnalités suivantes :

- définition des étapes initiales,
- définition des durées de diagnostic,
- commentaires sur l'étape,
- affectation d'actions et de leurs identificateurs.

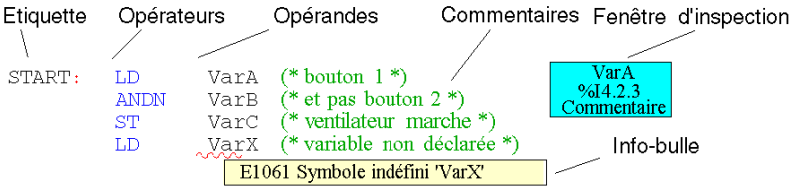
Liste d'instructions IL

Introduction

L'éditeur IL permet de programmer des listes d'instructions conformément à la norme CEI 61131-3.

Représentation

Représentation d'une section IL :



Objets

Une liste d'instructions se compose d'une chaîne d'instructions.

Chaque instruction commence dans une nouvelle ligne et se compose :

- d'un opérateur,
- éventuellement d'un modificateur,
- si nécessaire, d'un ou de plusieurs opérandes,
- éventuellement d'une étiquette servant de cible de saut,
- éventuellement d'un commentaire de la logique.

Aides à la saisie

L'éditeur IL propose entre autres les aides à la saisie suivantes :

- vérification sémantique et syntaxique dès la création du programme,
  - représentation en couleur des mots-clés et commentaires,
  - identification par un trait ondulé rouge des mots inconnus (p. ex. variables non déclarées) ou des types de données ne correspondant pas,
  - brève description de l'erreur dans l'info-bulle.
- affichage des fonctions et blocs fonction sous forme de tableau,
- aide à la saisie pour les fonctions et blocs fonction,
- possibilité de saisir et d'afficher les opérandes sous forme d'icône ou d'adresse topologique,
- affichage des fenêtres d'inspection.

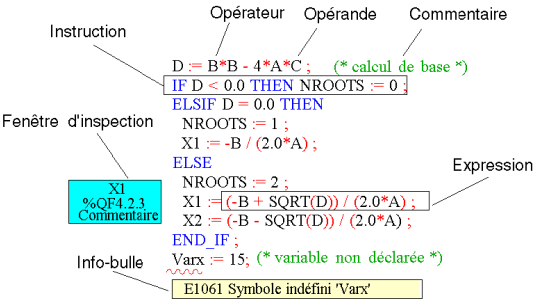
Littéral structuré ST

Introduction

L'éditeur ST permet la programmation en langage littéral structuré conformément à la norme CEI 61131-3.

Représentation

Représentation d'une section ST :



Objets

Le langage ST utilise ce que l'on appelle des "expressions".

Les expressions sont des constructions comprenant opérateurs et opérandes qui livrent une valeur lors de leur exécution.

Les opérateurs sont des symboles pour les opérations à exécuter.

Les opérateurs sont utilisés sur les opérandes. Les opérandes sont p. ex. des variables, des valeurs littérales, des entrées/sorties de fonction et bloc fonction, etc.

Les instructions servent à structurer et à commander les expressions.

Aides à la saisie

L'éditeur ST propose entre autres les aides à la saisie suivantes :

- vérification sémantique et syntaxique dès la création du programme,
  - représentation en couleur des mots-clés et commentaires,
  - identification par un trait ondulé rouge des mots inconnus (p. ex. variables non déclarées) ou des types de données ne correspondant pas,
  - brève description de l'erreur dans l'info-bulle.
- affichage des fonctions et blocs fonction sous forme de tableau,
- aide à la saisie pour les fonctions et blocs fonction,
- possibilité de saisir et d'afficher les opérandes sous forme d'icône ou d'adresse topologique,
- affichage des fenêtres d'inspection.

Simulateur automate

Introduction

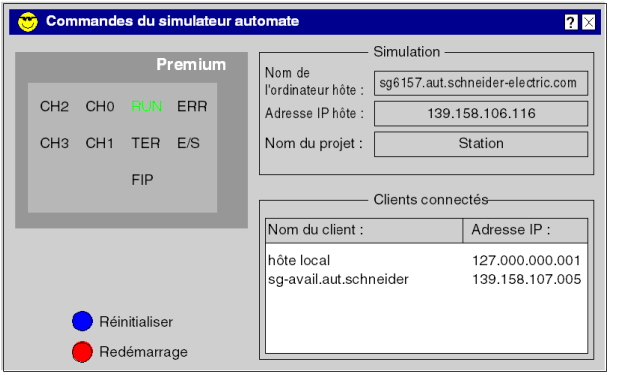
Le simulateur automate permet la recherche d'erreurs dans le projet sans connexion à un véritable automate.

Toutes les tâches du projet (Mast, Fast, AUX et Evénements) se déroulant sur un véritable automate sont également disponibles dans le simulateur. La différence par rapport à un véritable automate réside dans l'absence de modules E/S et de réseaux de communication (p. ex. ETHWAY, Fipio et Modbus Plus) fonctionnant en temps réel non-déterministe.

Naturellement, toutes les fonctions de mise au point, d'animation, les points d'arrêt, les forçages de variables, etc. sont disponibles sur le simulateur automate.

Représentation

Représentation de la boîte de dialogue :



Structure du simulateur

Le panneau du simulateur propose les affichages suivants :

- type d'automates simulés ;
- état actuel des automates simulés ;
- nom du projet chargé ;
- adresse IP et nom DNS du PC hôte du simulateur et de tous les PC Clients connectés ;
- boîte de dialogue dédiée à la simulation des événements E/S ;
- bouton **RAZ** permettant de réinitialiser les automates simulés (simulation de démarrage à froid) ;
- bouton **Mise sous/hors tension** (pour simuler une reprise à chaud) ;
- menu contextuel (bouton droit de la souris) permettant de commander le simulateur.

Exportation/Importation

Introduction

Les fonctions d'exportation et importation permettent d'utiliser dans un nouveau projet des données déjà créées. Le format d'exportation/importation XML permet en outre facilement de préparer des données d'un logiciel externe ou de les appliquer (copier).

Exportation

Les objets suivants peuvent être exportés :

- projets complets, configuration comprise,
- sections de tous les langages de programmation,
- sections de sous-programme de tous les langages de programmation,
- blocs fonctions dérivés (DFB),
- types de données dérivés (DDT),
- déclarations de variables,
- fenêtre utilisateur.

Importation

Il va de soi que tous les objets exportables peuvent également être réimportés.

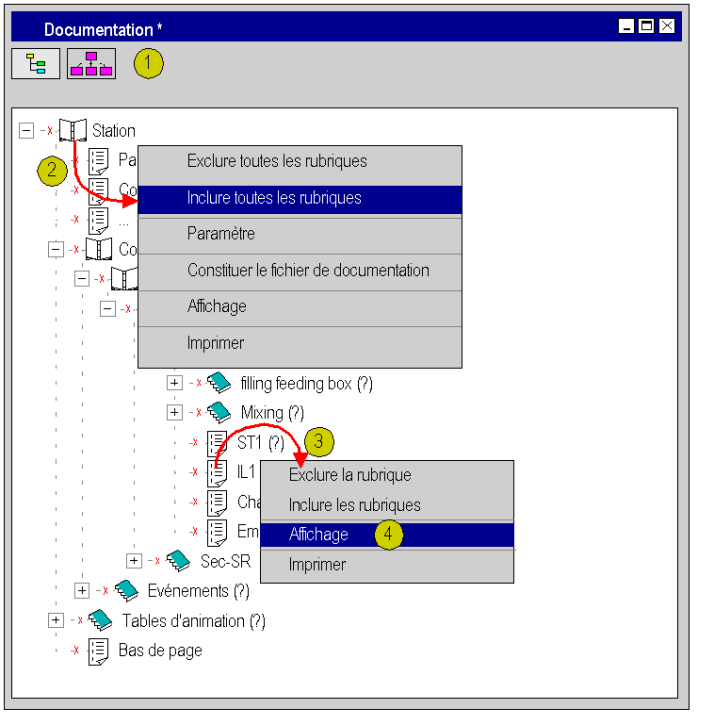
Il existe deux variantes d'importation :

- **importation directe**
  Importe l'objet exactement comme il a été exporté.
- **importation avec l'assistant**
  L'assistant permet de modifier les noms des variables, sections ou unités fonctionnelles. L'affectation des adresses peut également être modifiée.

Documentation utilisateur

Documentation utilisateur

Fonctions assurées par la documentation utilisateur :



Les fonctions suivantes sont entre autres disponibles pour la documentation du projet :

- impression de tout (2) ou partie (3) du projet
- sélection entre affichage structurel et fonctionnel (1)
- ajustement du résultat (bas de page, informations générales, etc.)
- impression locale pour les éditeurs de langage de programmation, le configurateur, etc.
- mise en évidence spéciale (écriture en gras) des mots clés
- libre choix du format de papier
- aperçu avant impression (4)
- enregistrement de la documentation

Services de mise au point

Recherche d'erreurs dans l'application utilisateur

Vous disposez notamment des fonctions suivantes pour optimiser la recherche d'erreurs dans le projet :

- pose de points d'arrêt (Break points) dans les éditeurs de langage de programmation,
- exécution pas à pas du programme (Step-by-step), avec pas à pas entrant (Step into), pas à pas sortant (Step out) et pas à pas principal (Step over),
- mémoire d'appel pour obtenir le chemin de programme complet,
- commande d'entrées et sorties.

Mode en ligne

Lorsque le PC est relié à l'automate et que la liaison est établie, on parle de mode en ligne.

Le mode en ligne est utilisé pour la recherche d'erreurs (mise au point), l'animation et la modification du programme sur l'automate.

Si la liaison doit être établie, une comparaison s'effectue automatiquement entre le projet du PC et l'automate.

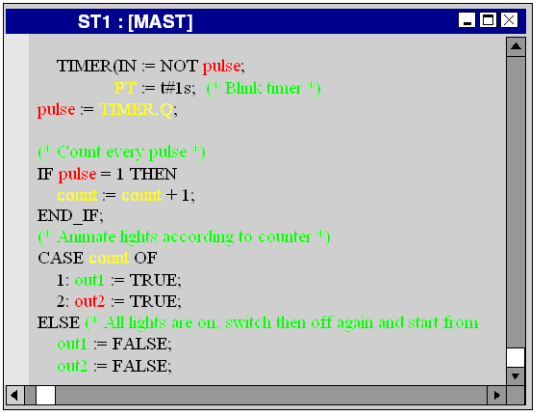
Cette comparaison peut donner les résultats suivants :

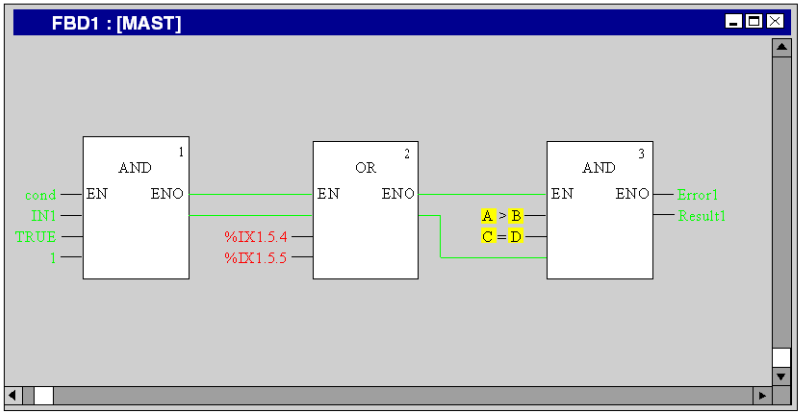
- **Projets différents sur le PC et sur l'automate**
  En pareil cas, le mode en ligne n'est disponible que de façon limitée. Seuls sont possibles les instructions de commande de l'automate (p. ex. démarrage, arrêt), les DiagServices (services de diagnostic) et le contrôle des variables. Il n'est pas possible de modifier la logique de programme ou la configuration sur l'automate. Les fonctions de chargement et de lecture sont également possibles et permettent de passer dans un mode illimité (projet identique sur le PC et sur l'automate).
- **Projets identiques sur le PC et sur l'automate**
  Deux possibilités se présentent :
  - **EN LIGNE IDENTIQUE**, **GENERE**
    La dernière génération du projet sur le PC a été chargée dans l'automate et aucune modification n'a ensuite été effectuée , en d'autres termes, le projet sur le PC et celui sur l'automate sont absolument identiques.
    Dans ce cas, toutes les fonctions d'animation sont disponibles sans restriction.
  - **EN LIGNE IDENTIQUE**, **NON GENERE**
    La dernière génération du projet sur le PC a été chargée dans l'automate, mais des modifications ont ensuite été effectuées.
    Dans ce cas, les fonctions d'animation ne sont disponibles que dans les parties du projet non modifiées.

Animation

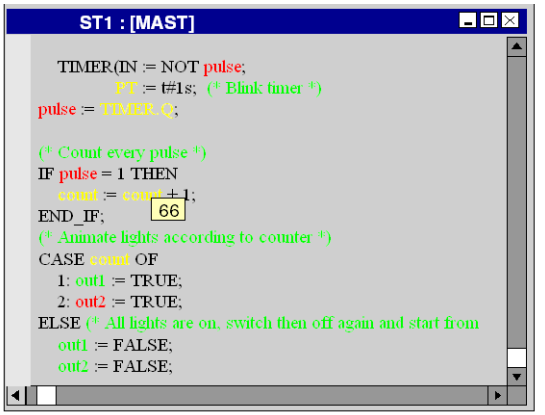
Plusieurs possibilités sont disponibles pour l'animation de variables :

- **Animation de section**
  Tous les langages de programmation (FBD, LD, SFC, IL et ST) peuvent être animés.
  L'animation des variables et des liaisons s'effectue directement dans la section.

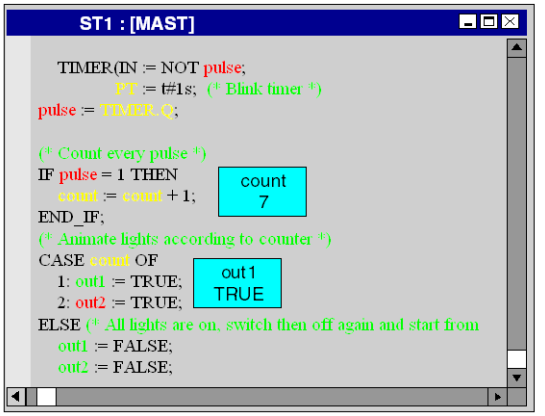




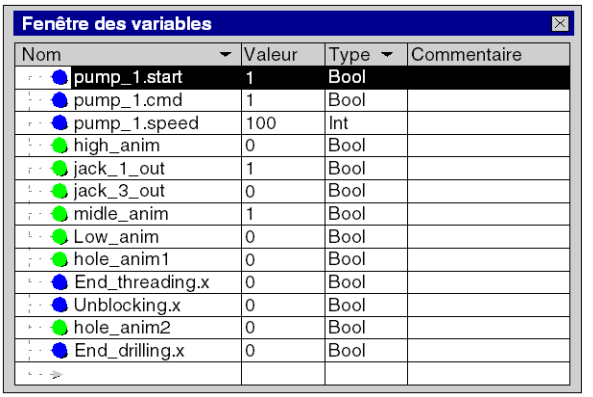
- **Info-bulles**
  Si le pointeur de la souris est placé sur une variable analogique, une info-bulle indique la valeur de celle-ci.



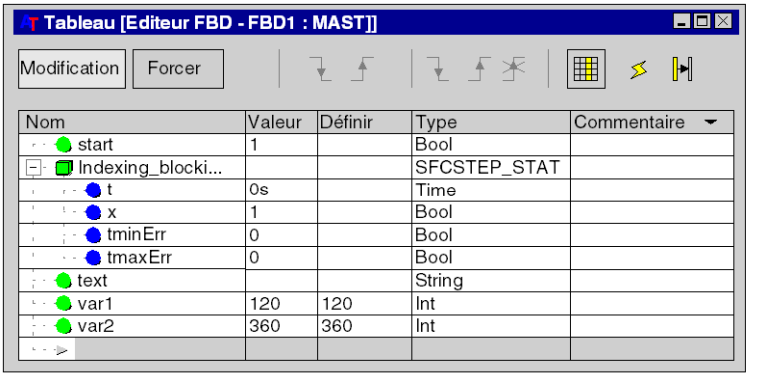
- **fenêtre de contrôle**
  Pour chaque variable, il est possible de créer une fenêtre de contrôle. Cette fenêtre affiche la valeur de la variable, son adresse et son commentaire (le cas échéant). Cette fonction est disponible dans tous les langages de programmation.



- **Fenêtre des variables**
  Cette fenêtre affiche toutes les variables utilisées dans la section courante.



- **Table d'animation**
  Dans les tables d'animation, il est possible d'afficher, modifier ou forcer les valeurs de toutes les variables du projet. Une ou plusieurs valeurs peuvent être modifiées simultanément.



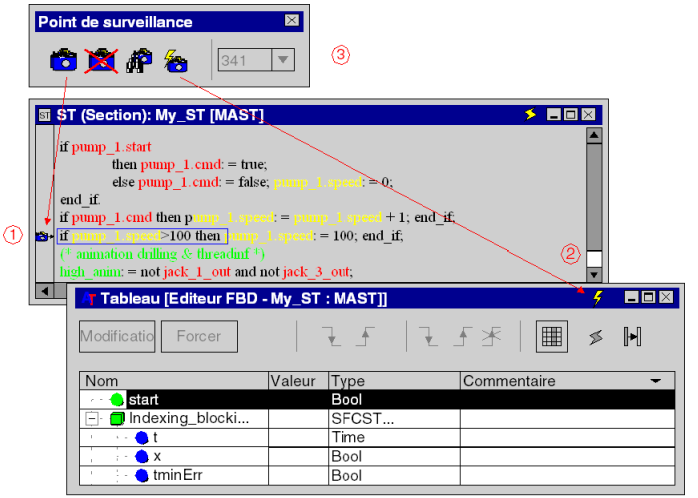
Point de surveillance

Les points de surveillance vous permettent de voir les données de l'automate exactement au moment de leur création (1) et pas seulement à la fin du cycle.

Les tables d'animation peuvent être synchronisées avec le point de surveillance (2).

Un compteur (3) indique la fréquence à laquelle le point de surveillance est actualisé.

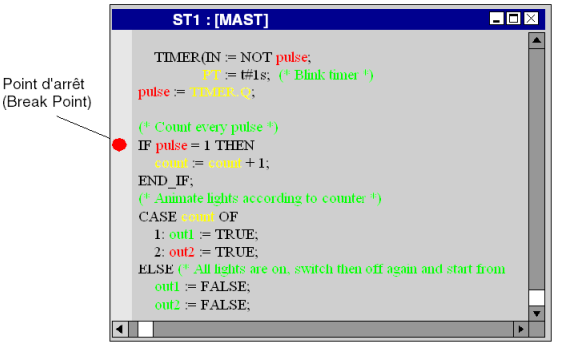
Section ST avec point de surveillance :



Point d'arrêt (Break Point)

Les points d'arrêt vous permettent d'arrêter l'exécution du projet à un endroit voulu.

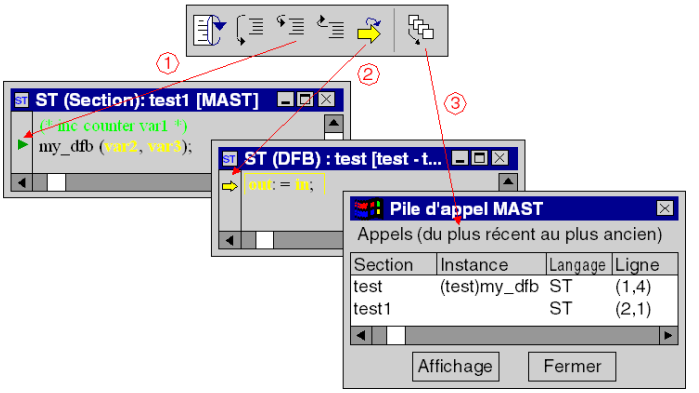
Section ST avec point d'arrêt :



Mode pas à pas

Le mode pas à pas vous permet d'exécuter le programme progressivement. Les fonctions pas à pas sont disponibles si le projet a été stoppé du fait de l'arrivée à un point d'arrêt ou s'il se trouve déjà en mode pas à pas.

Section ST en mode pas à pas :



Les fonctions suivantes sont disponibles en mode pas à pas :

- exécution progressive du programme (step-by-step),
- pas à pas entrant (step into) (1),
- pas à pas sortant (step out),
- pas à pas principal (step over),
- affichage de l'étape en cours d'exécution (2),
- mémoire d'appel (3).
  En cas d'exécution répétée de la fonction « pas à pas entrant » (step into), la mémoire d'appel permet d'afficher le chemin complet à partir du premier point d'arrêt (break point).

NOTE : l'exécution du programme de l'automate en mode pas à pas ainsi que la saisie (pas à pas entrant) dans une section protégée en lecture/écriture peut rendre impossible la lecture du programme et la fermeture de la section. L'utilisation doit basculer en mode « Stop » pour revenir à l'état initial.

Signets

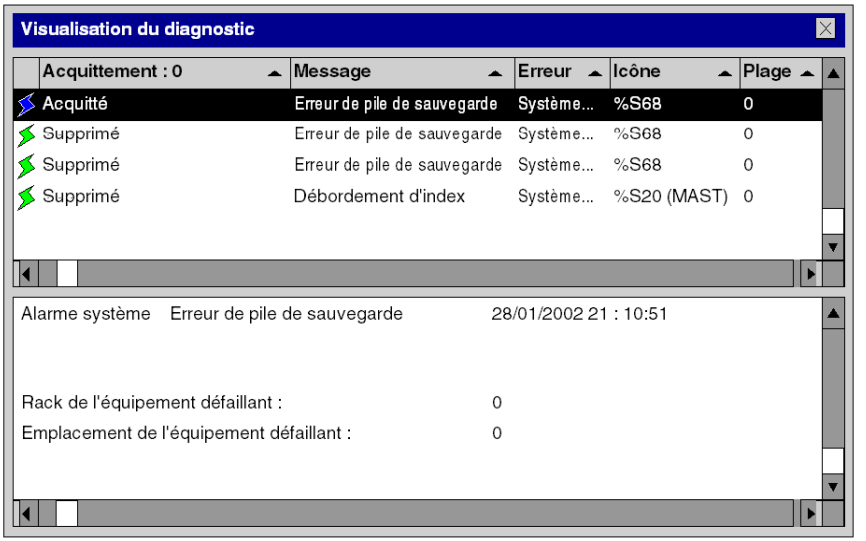
Les signets vous permettent de marquer des sections de code afin de les retrouver facilement.

Visualisation du diagnostic

Description

Unity Pro dispose d'un diagnostic du système et des projets.

Dans le cas où des erreurs se produisent, celles-ci s'affichent dans une fenêtre de diagnostic. Pour corriger l'erreur, il est possible d'ouvrir la section à l'origine de l'erreur directement depuis la fenêtre de visualisation du diagnostic.



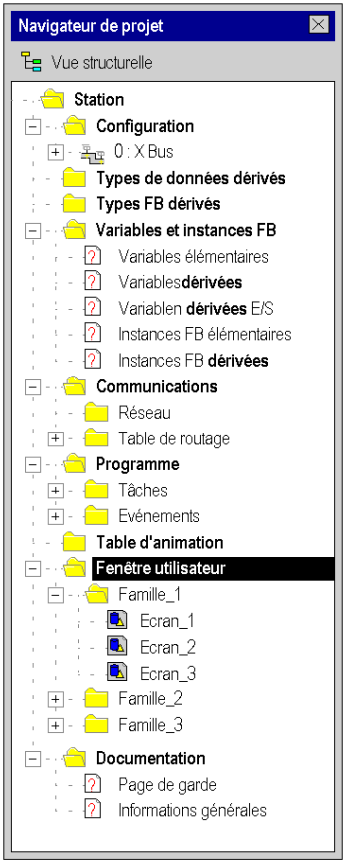
Fenêtre utilisateur

Introduction

Les fenêtres utilisateur permettent de visualiser le processus d'automatisation.

L'éditeur de fenêtres utilisateur permet de créer, modifier et gérer facilement des fenêtres utilisateur.

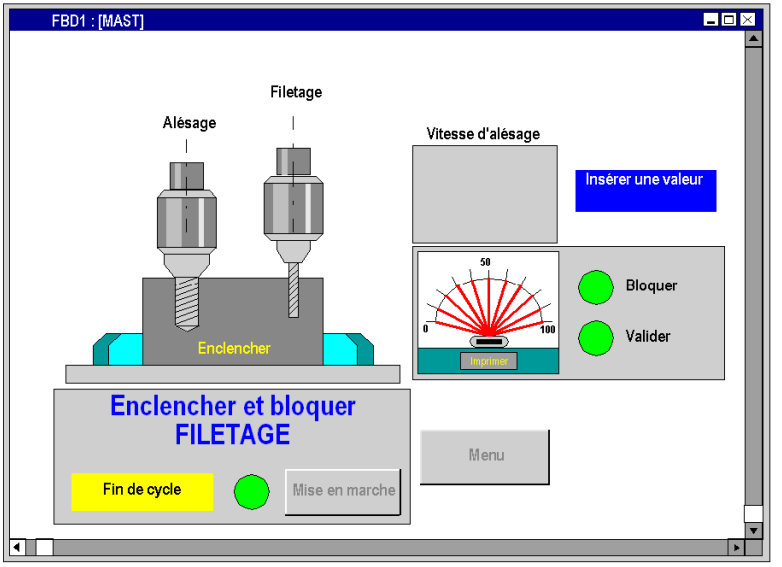
La création et l'accès aux fenêtres utilisateur s'effectuent via le navigateur de projet.



Editeur de fenêtres utilisateur

Une fenêtre utilisateur contient une foule d'informations (variables dynamiques, aperçus, textes descriptifs, ...) et permet de contrôler et modifier facilement et rapidement les variables d'automatisation.

Fenêtre utilisateur



L'éditeur de fenêtres utilisateur propose les fonctions suivantes :

- fonctions de visualisation complètes
  - éléments géométriques
    Ligne, rectangle, ellipse, courbe, polygone, bitmap, texte
  - éléments de commande (contrôles)
    Bouton, case à cocher, curseur, navigation à l'écran, liens hypertexte, zone de saisie, zone de sélection numérique
  - éléments d'animation
    Bargraphe, chronogramme, boîte de dialogue, apparition, disparition, couleurs clignotantes, animation de variables
- création d'une bibliothèque dédiée à la gestion des objets graphiques
- copie d'objets
- création d'une liste de toutes les variables utilisées dans la fenêtre utilisateur
- création de messages utilisés dans les fenêtres utilisateur
- accès direct à partir des fenêtres utilisateur à la table d'animation ou à la table de renvoi d'une ou de plusieurs variables
- info-bulles fournissant des informations complémentaires sur les variables
- gestion des fenêtres utilisateur par familles
- importation/exportation de certaines fenêtres utilisateur ou de familles complètes

Structure de l'application

II

Dans cette partie

Cette partie décrit les structures programme application et mémoire associées à chaque type d'automate.

Contenu de cette partie

Cette partie contient les chapitres suivants :

Description des fonctions disponibles pour chaque type d'automate

2

Fonctionnalités disponibles sur les différents types d'automates

Langages de programmation

Tous les langages suivants sont disponibles pour les plates-formes Modicon M340, Premium, Atrium et Quantum :

- LD
- FBD
- ST
- IL
- SFC

NOTE : Seuls les langages LD et FBD sont disponibles sur les modules de sécurité Quantum.

Tâches et processus

Le tableau suivant décrit les tâches et processus disponibles.

Plates-formes	**Modicon M340**		**Premium : TSX**			**Atrium : TSX**	**Quantum : 140 CPU**		
Processeurs	P34 1000	P34 20••	P57 0244 P57 1••	P57 2•• P57 3•• P57 4•• H57 24M H57 44M	P57 5•• P57 6634	PCI 57 204/354	31•••• 43•••• 53••••	651•• 652 60 671 60	651 60S 671 60S
Tâche maître cyclique ou périodique	X	X	X	X	X	X	X	X	X
Tâche rapide périodique	X	X	X	X	X	X	X	X	-
Tâches auxiliaires périodique	-	-	-	-	4	-	-	4	-
Taille maximum d'une section			64 Ko					16 Mo	-
Traitement événementiel type E/S	32	64	32	64	128	64	64	128	-
Traitement événementiel type Timer	16	32	-	-	32	-	16	32	-
Total des traitements événementiels type E/S + Timer	32	64	32	64	128	64	64	128	-

X ou valeur tâches ou processus disponibles (la valeur correspond au nombre maximum).

- tâches ou traitements indisponibles.

Structure du programme d'application

3

Objet de ce chapitre

Ce chapitre décrit la structure et l'exécution des programmes réalisés à l'aide du logiciel Unity Pro.

Contenu de ce chapitre

Ce chapitre contient les sous-chapitres suivants :

3.1 Description des tâches et des traitements

Objet de cette section

Ce sous-chapitre décrit les tâches et les traitements composant le programme application.

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Présentation de la tâche maître

Généralités

La tâche maître représente la tâche principale du programme d'application. Elle est obligatoire et créée par défaut.

Structure

La tâche maître (MAST) est composée de sections et sous-programmes.

Chaque section de la tâche maître est programmée dans les langages suivants : LD, FBD, IL, ST ou SFC.

Les sous-programmes sont programmés en langage LD, FBD, IL ou ST et sont appelés dans les sections de tâches.

NOTE : SFC ne sont utilisables que dans les sections de tâche maître. Le nombre de sections programmées dans SFC est illimité.

Exécution

L'exécution de la tâche maître peut être choisie :

- cyclique (sélection par défaut)
- ou périodique (1 à 255 ms)

Contrôle

La tâche maître peut être contrôlée par le programme, les bits et les mots système.

Objets système	Description
%SW0	Période de tâche.
%S30	Activation de la tâche maître.
%S11	Erreur du chien de garde.
%S19	Dépassement période.
%SW27	Durée du dernier cycle Mast (en millisecondes).
%SW28	Durée maximum du surdébit (en ms) pour Modicon M340.
%SW29	Durée minimum du surdébit (en ms) pour Modicon M340.
%SW30	Temps d'exécution (en ms) du dernier cycle.
%SW31	Temps d'exécution (en ms) du cycle le plus long.
%SW32	Temps d'exécution (en ms) du cycle le plus court.

Présentation de la tâche rapide

Généralités

La tâche rapide est destinée aux traitements de courte durée et périodiques.

Constitution

La tâche rapide (FAST) est constituée de sections, et de sous-programmes.

Chaque section de la tâche rapide se programme dans un des langages : LD, FBD, IL ou ST.

Le langage SFC n'est pas utilisable dans les sections d'une tâche rapide.

Les sous-programmes se programment en langage LD, FBD, IL ou ST, ils sont appelés dans les sections de la tâche.

Exécution

L'exécution de la tâche rapide est périodique.

Elle est plus prioritaire que la tâche maître.

La période de la tâche rapide (FAST) est fixée par configuration, de 1 à 255 ms.

Le programme exécuté doit cependant rester court pour éviter le débordement des tâches moins prioritaires.

Contrôle

La tâche rapide peut être contrôlée par programme par bits et mots système.

Objets système	Description
%SW1	Période de tâche.
%S31	Activation de la tâche rapide.
%S11	Erreur du chien de garde
%S19	Dépassement période.
%SW33	Temps d'exécution (en ms) du dernier cycle.
%SW34	Temps d'exécution (en ms) du cycle le plus long.
%SW35	Temps d'exécution (en ms) du cycle le plus court.

Présentation des tâches auxiliaires

Généralités

Les tâches auxiliaires sont conçues pour les tâches de traitement plus lentes. Ces tâches ont la priorité la plus faible.

Il est possible de programmer jusqu'à 4 tâches auxiliaires (AUX0, AUX1, AUX2 ou AUX3) sur les automates Premium TSX P57 5•• et Quantum 140 CPU 6••••. Les tâches auxiliaires ne sont pas disponibles pour les automates Modicon M340.

Structure

Les tâches auxiliaires (AUX) sont composées de sections et sous-programmes.

Chaque section de la tâche auxiliaire est programmée dans l'un des langages suivants : LD, FBD, IL ou ST.

Le langage SFC n'est pas utilisable dans les sections d'une tâche auxiliaire.

Un maximum de 64 sous-programmes peuvent être programmés en langage LD, FBD, IL ou ST. Ils sont appelés dans les sections de tâches.

Exécution

L'exécution des tâches auxiliaires est périodique.

Elles ont la priorité la plus faible.

La période de la tâche auxiliaire peut être fixée entre 10 ms et 2,55 s.

Contrôle

Les tâches auxiliaires peuvent être contrôlées par le programme, les bits et les mots système.

Objets système	Description
%SW2	Période de la tâche auxiliaire 0
%SW3	Période de la tâche auxiliaire 1
%SW4	Période de la tâche auxiliaire 2
%SW5	Période de la tâche auxiliaire 3
%S32	Activation de la tâche auxiliaire 0
%S33	Activation de la tâche auxiliaire 1
%S34	Activation de la tâche auxiliaire 2
%S35	Activation de la tâche auxiliaire 3
%S11	Erreur du chien de garde
%S19	Dépassement période.
%SW36	Temps d'exécution du dernier cycle d'exécution de la tâche auxiliaire 0 (en ms).
%SW39	Temps d'exécution du dernier cycle d'exécution de la tâche auxiliaire 1 (en ms).
%SW42	Temps d'exécution du dernier cycle d'exécution de la tâche auxiliaire 2 (en ms).
%SW45	Temps d'exécution du dernier cycle d'exécution de la tâche auxiliaire 3 (en ms).
%SW37	Temps d'exécution du cycle d'exécution de la tâche auxiliaire le plus long (en ms).
%SW40	Temps d'exécution du cycle d'exécution de la tâche auxiliaire 1 le plus long (en ms).
%SW43	Temps d'exécution du cycle d'exécution de la tâche auxiliaire 2 le plus long (en ms).
%SW46	Temps d'exécution du cycle d'exécution de la tâche auxiliaire 3 le plus long (en ms).
%SW38	Temps d'exécution du cycle d'exécution de la tâche auxiliaire 0 le plus court (en ms).
%SW41	Temps d'exécution du cycle d'exécution de la tâche auxiliaire 1 le plus court (en ms).
%SW44	Temps d'exécution du cycle d'exécution de la tâche auxiliaire 2 le plus court (en ms).
%SW47	Temps d'exécution du cycle d'exécution de la tâche auxiliaire 3 le plus court (en ms).

Gestions des traitements événementiels

Généralités

Le traitement de l'événement est utilisé pour réduire le temps de réponse du programme d'application aux événements :

- provenant de modules d'entrées/sorties,
- des événements Timer.

Ces tâches de traitement sont prioritaires sur toutes les autres tâches. Elles conviennent donc aux tâches de traitement demandant des temps de réponse très courts par rapport à l'événement.

Le nombre de tâches de traitement de l'événement *(voir page 66)* qui peut être programmé dépend du type de processeur.

Structure

Une tâche de traitement de l'événement est composée d'une section unique (non conditionnée).

Elle est programmée dans le langage LD, FBD, IL ou ST.

Deux types d'événements sont proposés :

- Evénement E/S : pour les événements provenant de modules d'entrées/sorties
- Evénement TIMER : pour les événements provenant d'événements Timer

Exécution

L'exécution d'une tâche de traitement de l'événement est asynchrone.

L'occurrence d'un événement achemine le programme d'application à la tâche de traitement associée à la voie d'entrée et de sortie ou l'événement Timer qui a causé l'événement.

Contrôle

Les bits et mots système suivants peuvent être utilisés pour contrôler les tâches de traitement de l'événement lors de l'exécution du programme.

Objets système	Description
%S38	Activation de traitement de l'événement.
%S39	Saturation de la pile de gestion des appels d'événements.
%SW48	Nombre de tâches de traitement de l'événement exécutées.
%SW75	Nombre des événements de type Timer de la file d'attente.

3.2 Description de sections et de sous-programmes

Objectif de cette section

La section suivante décrit les sections et sous-programmes qui constituent la tâche.

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Description de sections

Présentation des sections

Les sections sont des entités autonomes de programmation.

Les étiquettes de repérage des lignes d'instructions, des réseaux de contacts ... sont propres à la section (pas de saut de programme possible vers une autre section).

Elles se programment soit en :

- langage à contacts LD,
- langage en blocs fonctionnels FBD,
- Liste d'instructions IL,
- littéral structuré ST,
- diagramme fonctionnel en séquence SFC,

sous réserve que le langage soit accepté dans la tâche.

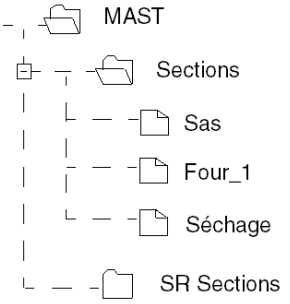
Les sections sont exécutées dans leur ordre de programmation dans la fenêtre du navigateur (vue structurelle).

Une condition d'exécution peut être associée à une ou plusieurs sections dans les tâches maître, rapides et auxiliaires, mais pas dans les traitements événementiels.

Elles sont reliées à une tâche. Une même section ne peut pas appartenir à plusieurs tâches en même temps.

Exemple

Le dessin suivant donne l'exemple de structure d'une tâche en sections.



Caractéristiques d'une section

Le tableau suivant décrit les caractéristiques d'une section.

Caractéristique	Description
Nom	32 caractères maximum (les accents sont possibles, mais les espaces sont interdits).
Langage	LD, FBD, IL, ST ou SFC
Tâche ou traitement	Maître, rapide, auxiliaires, événéméntiel
Condition (optionnel)	Une variable bit de type BOOL ou EBOOL peut être utilisée pour conditionner l'exécution de la section.
Commentaire	256 caractères maximum.
Protection	Protection en écriture, protection en lecture/écriture.

Description des sections SFC

Généralités

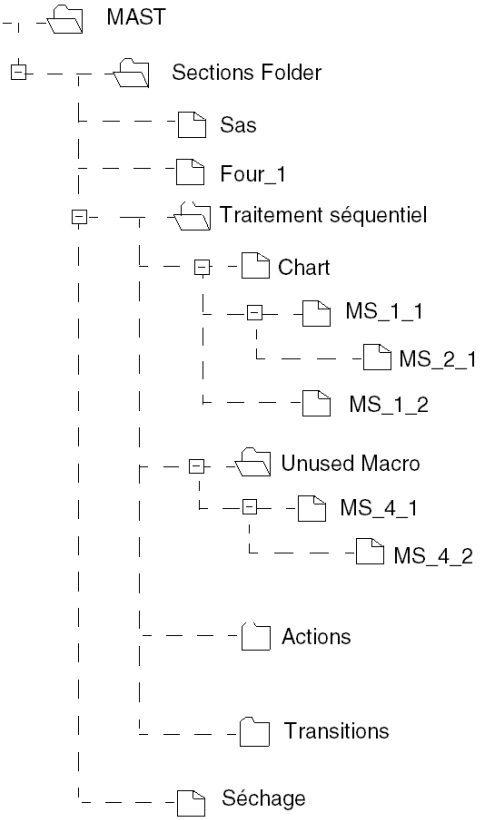
Les sections en langage diagramme fonctionnel en séquence se composent :

- d'un graphe principal programmé en SFC,
- de macro-étapes (MS) programmés en SFC
- d'actions et transitions programmées en LD, FBD , ST ou IL

Les sections SFC sont programmables uniquement dans la tâche maître (voir description détaillée des sections SFC).

Exemple

Le diagramme suivant donne un exemple de constitution d'une section SFC et montre les appels des macro-étapes utilisées à partir du diagramme.



Description de sous-programmes

Présentation des sous-programmes

Les sous-programmes sont programmés comme des entités distinctes en :

- langage à contacts LD,
- langage en blocs fonctionnels FBD,
- liste d'instructions IL,
- Littéral structuré (ST).

Les appels aux sous-programmes s'effectuent à partir des sections ou d'un autre sous-programme.

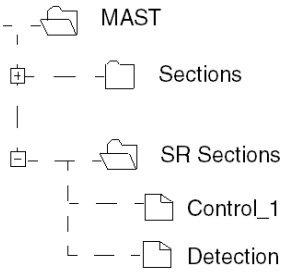
Le nombre d'imbrications est limité à 8.

Un sous-programme ne peut pas s'appeler lui-même (non récursif).

Les sous-programmes sont également liés à une tâche. Un même sous-programme ne peut pas être appelé par différentes tâches.

Exemple

Le dessin suivant donne l'exemple de structure d'une tâche en sections et en sous-programmes.



Caractéristiques d'un sous-programme

Le tableau suivant décrit les caractéristiques d'un sous-programme.

Caractéristique	Description
Nom	32 caractères maximum (les accents sont possibles, mais les espaces sont interdits).
Langage	LD, FBD, IL ou ST.
Tâche	Maître, rapide ou auxiliaire
Commentaire	512 caractères maximum.

3.3 Exécution monotâche

Objet de cette section

Cette section décrit le fonctionnement d'une application monotâche.

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Description du cycle de tâche maître

Généralités

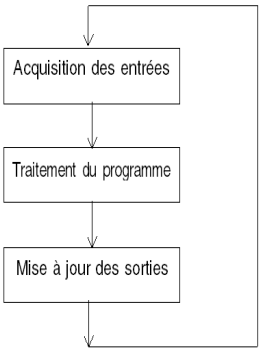
Le programme d'une application monotâche est associé à une seule tâche utilisateur la tâche maître *(voir page 69)*.

L'exécution de la tâche maître peut être choisie :

- cyclique
- périodique

Illustration

L'illustration suivante montre le cycle de fonctionnement.



Description des différentes phases

Le tableau ci-après décrit les phases de fonctionnement.

Phase	Description
Acquisition des entrées	Ecriture en mémoire de l'état des informations présentes sur les entrées des modules TOR et métier associées à la tâche, Ces valeurs peuvent être modifiées par les valeurs de forçage.
Traitement du programme	Exécution du programme application, écrit par l'utilisateur,
Mise à jour des sorties	Ecriture des bits ou des mots de sorties associés aux modules TOR et métier associés à la tâche selon l'état défini par le programme application. Comme pour les entrées, l'écriture des sorties peut être modifiée par les valeurs de forçage.

NOTE : Durant les phases d'acquisition des entrées et de mise à jour des sorties, le système réalise aussi implicitement la surveillance de l'automate (gestion des bits et mots système, mise à jour des valeurs courantes de l'horodateur, mise à jour des voyants d'état LED et écrans LCD (par pour Modicon M340), détection des passages RUN/STOP, ...) et le traitement des requêtes en provenance du terminal (modifications et animation).

Mode de marche

Automate en RUN, le processeur effectue dans l'ordre le traitement interne, l'acquisition des entrées, le traitement du programme application et la mise à jour des sorties.

Automate en STOP, le processeur effectue :

- le traitement interne,
- l'acquisition des entrées (1),
- et suivant la configuration choisie :
  - mode repli : les sorties sont mises en position de "repli",
  - mode maintien : les sorties sont maintenues à leur dernière valeur.

(1) dans le cas des automates Premium, Atrium et Quantum, l'acquisition des entrées est inhibée lorsque l'automate est en STOP.

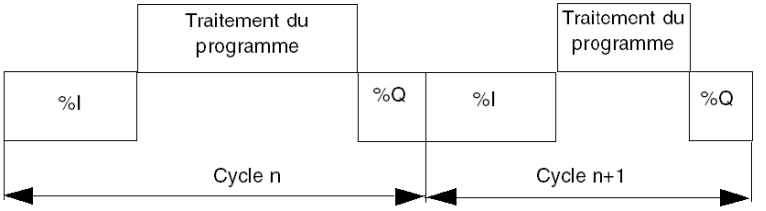
Monotâche : Exécution cyclique

Généralités

La tâche maître fonctionne comme suit. Une description de l'exécution cyclique de la tâche maître est fournie pour le fonctionnement monotâche.

Fonctionnement

L'illustration suivante montre les phases d'exécution du cycle automate.



%I Lecture des entrées
%Q Ecriture des sorties

Description

Ce type de fonctionnement consiste à enchaîner les cycles de tâche les uns après les autres.

Après la mise à jour des sorties, le système effectue sont propre traitement et commence un autre cycle de tâche, sans s'arrêter.

Contrôle du cycle

Le cycle est contrôlé par le chien de garde *(voir page 84)*.

Exécution périodique

Description

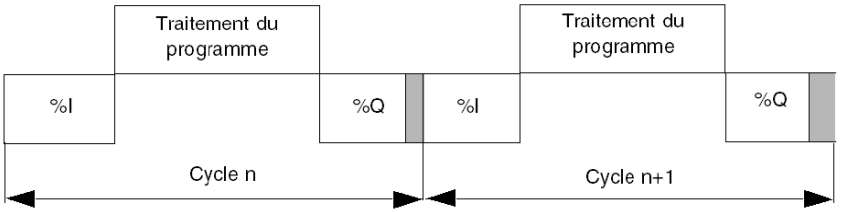
Dans ce mode de fonctionnement, l'acquisition des entrées, le traitement du programme application et la mise à jour des sorties s'effectuent de façon périodique selon une période définie de 1 à 255 ms.

En début de cycle automate, une temporisation dont la valeur courante est initialisée à la période définie, commence à décompter.

Le cycle automate doit se terminer avant l'expiration de cette temporisation qui relance un nouveau cycle.

Fonctionnement

Le dessin suivant montre les phases d'exécution du cycle automate.



%I Lecture des entrées
%Q Ecriture des sorties

Mode de marche

Le processeur effectue dans l'ordre le traitement interne, l'acquisition des entrées, le traitement du programme application et la mise à jour des sorties.

- Si la période n'est pas encore terminée, le processeur complète son cycle de fonctionnement jusqu'à la fin de la période par du traitement interne.
- Si le temps de fonctionnement devient supérieur à celui affecté à la période, l'automate signale un débordement de période par la mise à l'état 1 du bit système %S19 de la tâche, le traitement se poursuit et est exécuté dans sa totalité (il ne doit pas dépasser néanmoins le temps limite du chien de garde). Le cycle suivant est enchaîné après l'écriture implicite des sorties du cycle en cours.

Contrôle du cycle

Deux contrôles sont effectués :

- débordement de période *(voir page 84)*,
- par chien de garde *(voir page 84)*.

Contrôle de la durée du cycle

Généralités

La période de l'exécution de la tâche maître, en fonctionnement cyclique ou périodique, est contrôlée par l'automate (chien de garde) et ne doit pas excéder la valeur définie dans la configuration Tmax (1500 ms par défaut, 1,5 s maximum).

Chien de garde logiciel (fonctionnement périodique ou cyclique)

Si le dépassement du chien de garde se produit, l'application est déclarée en erreur, ce qui entraîne l'arrêt immédiat de l'automate (état HALT).

Le bit %S11 indique que le chien de garde est dépassé. Il est défini sur 1 par le système lorsque la durée de cycle est supérieure au chien de garde.

Le mot %SW11 contient la valeur du chien de garde en ms. Cette valeur n'est pas modifiable par le programme.

NOTE :

- La réactivation de la tâche requiert la connexion du terminal afin d'analyser la cause de l'erreur, la corriger, réinitialiser l'automate et le faire passer sur RUN.
- Il n'est pas possible de quitter HALT en basculant vers STOP. Pour ce faire, vous devez réinitialiser l'application pour vérifier la cohérence des données.

Contrôle en fonctionnement périodique

En fonctionnement périodique, un contrôle supplémentaire permet la détection d'un dépassement période. Un dépassement période n'entraîne pas l'arrêt de l'automate tant qu'il reste inférieur à la valeur du chien de garde.

Le bit %S19 indique que la période est dépassée. Il est défini sur 1 par le système lorsque la durée de cycle est supérieure à la période de la tâche.

Le mot %SW0 contient la valeur de la période (en ms). Il est initialisé lors d'un redémarrage à froid par la valeur définie. Il peut être modifié par l'utilisateur.

Exploitation des temps d'exécution de tâche maître

Les mots système suivants peuvent être utilisés pour obtenir des informations sur la durée de cycle :

- %SW30 contient le temps d'exécution du dernier cycle
- %SW31 contient le temps d'exécution du cycle le plus long
- %SW32 contient le temps d'exécution du cycle le plus court

NOTE : Vous pouvez explicitement accéder à ces différentes informations à partir de l'éditeur de configuration.

Exécution des sections Quantum avec entrées/sorties décentralisées

Généralités

Les automates Quantum possèdent un système de gestion des sections spécifique. Il s'applique aux stations d'entrées/sorties décentralisées.

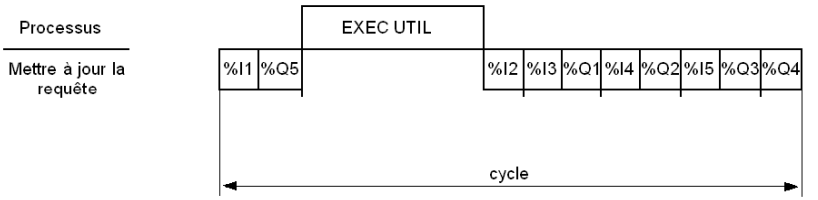
Ces stations sont utilisées avec les modules d'E/S distantes suivants :

- 140 CRA 931 00
- 140 CRA 932 00

Ce système permet une mise à jour des entrées/sorties décentralisées au niveau des sections assurant des temps de réaction optimums (sans attente du temps de cycle complet de la tâche pour la mise à jour des entrées/sorties).

Fonctionnement

Le schéma suivant présente les phases d'E/S lorsque 5 stations d'E/S sont associées à des sections de tâche client.



%Ii entrées de la station d'E/S n° i
%Qi sorties de la station d'E/S n° i
i numéro de station d'E/S

Description

Phase	Description
1	Requête de mise à jour : ● des entrées de la première station d'E/S (i=1) ● des sorties de la dernière station d'E/S (i=5)
2	Traitement du programme
3	● Mise à jour des entrées de la première station d'E/S (i=1) ● Requête de mise à jour des entrées de la deuxième station d'E/S (i=2)
4	Requête de mise à jour : ● des entrées de la troisième station d'E/S (i=3) ● des sorties de la première station d'E/S (i=1)
5	Requête de mise à jour : ● des entrées de la quatrième station d'E/S (i=4) ● des sorties de la deuxième station d'E/S (i=2)
6	Requête de mise à jour : ● des entrées de la dernière station d'E/S (i=5) ● des sorties de la troisième station d'E/S (i=3)
7	Requête de mise à jour des sorties de la quatrième station d'E/S (i=4)

Réglage du temps de maintien de la station

Afin que les sorties distantes soient correctement mises à jour et afin d'éviter que les valeurs de repli ne soient appliquées, le temps de maintien défini doit être au moins le double du temps de cycle de la tâche MAST. La valeur par défaut, 300 ms, doit donc être modifiée si la période MAST est réglée sur la valeur maximum de 255 ms. L'ajustement du temps de maintien de la station doit être effectué sur toutes les stations configurées.

3.4 Exécution multitâche

Objet de cette section

Cette section décrit le fonctionnement d'une application multitâche.

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Structure logicielle multitâche

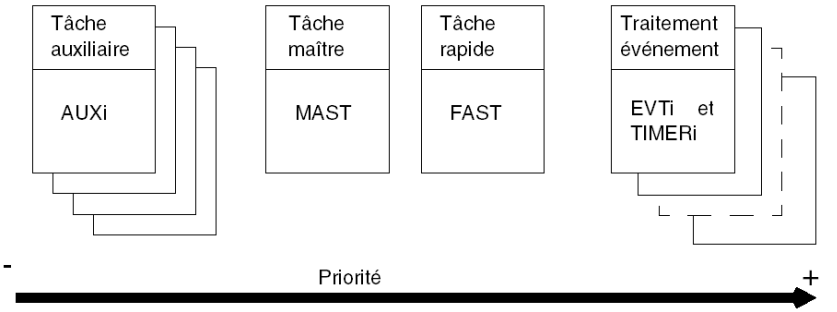
Tâches et traitements

La structure de tâche de ce type d'application est la suivante :

Tâche/traitement	Désignation	Description
Maître	MAST	Toujours présent, peut être défini sur cyclique ou périodique.
Fast	FAST	Facultatif, toujours périodique.
Auxiliaire	AUX 0 à 3	Facultatif et toujours périodique.
Evénement	EVTi et TIMERi *(voir page 97)*	Appelé par le système lorsqu'un événement se produit sur un module d'entrées/sorties ou est déclenché par l'événement TIMER. Ces types de traitements sont facultatifs et peuvent être utilisés par des applications devant agir sur les entrées et sorties dans un délai très court.

Illustration

La figure suivante indique les tâches d'une structure multitâche ainsi que leur niveau de priorité.



Description

La tâche maître (MAST) est toujours l'application de base. Les autres tâches varient en fonction du type d'automate *(voir page 66)*.

Des niveaux de priorité sont déterminés pour chaque tâche de façon à établir un ordre d'importance pour les types de traitements.

Le traitement événementiel peut être activé de façon asynchrone par rapport aux tâches périodiques, dans un ordre généré par des événements externes. Celui-ci est traité en priorité et nécessite l'interruption de tout autre traitement en cours.

Précautions

Multitâche : règles d'or

⚠ ATTENTION

COMPORTEMENT INATTENDU DE L'APPLICATION MULTITACHE

Le partage d'entrées et de sorties entre différentes tâches peut générer un dysfonctionnement de l'application.

Il est vivement recommandé d'associer chaque entrée ou sortie à une tâche uniquement.

Le non-respect de ces instructions peut provoquer des blessures ou des dommages matériels.

Séquencement des tâches dans une structure multitâche

Généralités

La tâche maître est active par défaut.

Les tâches rapide et auxiliaires sont actives par défaut si elles sont programmées.

Le traitement événementiel est activé lors d'apparition de l'événement qui lui a été associé.

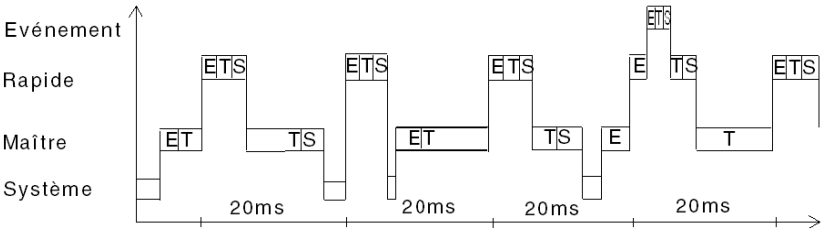
Fonctionnement

Le tableau suivant décrit l'exécution des tâches prioritaires (ce fonctionnement est aussi illustré par le dessin ci-dessous).

Phase	Description
1	Arrivée d'un événement ou début de cycle de la tâche rapide.
2	Arrêt de l'exécution des tâches en cours moins prioritaires,
3	Exécution de la tâche prioritaire.
4	La tâche interrompue reprend la main lorsque les traitements de la tâche prioritaire se termine.

Description du séquencement des tâches

Le diagramme suivant illustre le séquencement des tâches d'un traitement multitâche comportant une tâche maître cyclique, une tâche rapide de période 20 ms et un traitement événementiel.



Légende :

E : acquisition des entrées

T : traitement du programme

S : mise à jour des sorties

Contrôle des tâches

L'exécution des tâches rapide et événementielles peut être contrôlée par programme à travers l'utilisation des bits système :

- %S30 permet d'activer ou non la tâche maître MAST.
- %S31 permet d'activer ou non la tâche rapide FAST.
- %S32 à %S35 permettent d'activer ou pas les tâches auxiliaires AUX0 à AUX3.
- %S38 permet d'activer ou pas les traitements événementiels EVTi.

NOTE : Les fonctions élémentaires MASKEVT et UNMASKEVT permettent aussi le masquage et le démasquage global des événements par programme.

Contrôle des tâches

Fonctionnement cyclique ou périodique

En fonctionnement multitâche, la tâche avec la priorité la plus élevée devra être utilisée en mode périodique pour permettre aux tâches avec la priorité la plus basse de s'exécuter.

Pour cette raison, seule la tâche avec la priorité la plus faible devrait être utilisée en mode cyclique. Par conséquent, choisir le mode de fonctionnement cyclique pour la tâche maître exclut l'utilisation de tâches auxiliaires.

Mesure des durées des tâches

La durée des tâches est mesurée en continu. Cette mesure représente la durée entre le démarrage et la fin de l'exécution de la tâche. Elle inclut le temps mis par les tâches de priorité de niveau le plus élevé qui peuvent interrompre l'exécution de la tâche mesurée.

Les mots système suivants donnent les durées des cycles actuels, maximum et minimum pour chaque tâche (valeur en ms)

Mesure des durées	Cycle	FAST	AUX0	AUX1	AUX2	AUX3
Courant	%SW30	%SW33	%SW36	%SW39	%SW42	%SW45
Maximum	%SW31	%SW34	%SW37	%SW40	%SW43	%SW46
Durée	%SW32	%SW35	%SW38	%SW41	%SW44	%SW47

NOTE : les durées maximum et minimum sont prises à partir des durées mesurées depuis le dernier redémarrage à froid.

Périodes de tâche

Les périodes sont définies dans les propriétés de la tâche. Elles peuvent être modifiées par les mots système suivants.

Mots système	Tâche	Valeurs	Valeurs par défaut	Remarques
%SW0	Cycle	0-255 ms	Cyclique	0 = en fonctionnement cyclique
%SW1	FAST	1-255 ms	5 ms	-
%SW2	AUX0	10 ms-2,55 s	100 ms	Les valeurs de la période sont exprimées en 10 ms.
%SW3	AUX1	10 ms-2,55 s	200 ms	
%SW4	AUX2	10 ms-2,55 s	300ms	
%SW5	AUX3	10 ms-2,55 s	400ms	

Lorsque la durée de cycle de la tâche dépasse la période, le système règle le bit système %S19 de la tâche sur 1 et continue avec le cycle suivant.

NOTE : les valeurs des périodes ne dépendent pas de la priorité des tâches. Il est possible de définir la période d'une tâche rapide qui est plus importante que la tâche maître.

Chien de garde

L'exécution de chaque tâche est contrôlée par un chien de garde configurable, à l'aide des propriétés de la tâche.

Le tableau suivant offre la place des valeurs du chien de garde pour chacune des tâches.

Tâches	Valeurs du chien de garde (min...max) (ms)	Valeur du chien de garde par défaut (ms)	Mot système associé
Cycle	10-1500	250	%SW11
FAST	10..500	100	-
AUX0	100-5000	2000	-
AUX1	100-5000	2000	-
AUX2	100-5000	2000	-
AUX3	100-5000	2000	-

Si le dépassement du chien de garde se produit, l'application est déclarée en erreur, ce qui entraîne l'arrêt immédiat de l'automate (état HALT).

Le mot %SW11 contient la valeur du chien de garde de la tâche maître en ms. Cette valeur n'est pas modifiable par le programme.

Le bit %S11 indique que le chien de garde est dépassé. Il est réglé sur 1 par le système lorsque la durée de cycle est supérieure au chien de garde.

NOTE :

- la réactivation de la tâche requiert la connexion du terminal afin d'analyser la cause de l'erreur, la corriger, réinitialiser l'automate et le faire passer sur RUN.
- Il n'est pas possible de quitter HALT en basculant vers STOP. Pour ce faire, vous devez réinitialiser l'application pour vérifier la cohérence des données.

Contrôle des tâches

Lorsque le programme d'application s'exécute, il est possible d'activer ou d'inhiber une tâche à l'aide des bits système suivants :

Bits système	Tâche
%S30	Cycle
%S31	FAST
%S32	AUX0
%S33	AUX1
%S34	AUX2
%S35	AUX3

La tâche est active lorsque le bit système associé est réglé sur 1. Ces bits sont testés par le système à la fin de la tâche maître.

Lorsqu'une tâche est inhibée, les entrées continuent à être lues et les sorties à être écrites.

Au démarrage du programme d'application, uniquement au premier cycle d'exécution, la tâche maître est active. A la fin du premier cycle, les autres tâches sont automatiquement activées sauf si l'une des tâches est inhibée (associée au bit système réglé sur 0) par le programme.

Contrôles des phases de lecture d'entrée et d'écriture de sortie

Les bits des mots système suivants peuvent être utilisés (uniquement lorsque l'automate est en mode RUN) pour inhiber les phases de lecture d'entrée et d'écriture de sortie.

Inhibition de phases...	Cycle	FAST	AUX0	AUX1	AUX2	AUX3
lecture d'entrées	%SW8.0	%SW8.1	%SW8.2	%SW8.3	%SW8.4	%SW8.5
écriture des sorties	%SW9.0	%SW9.1	%SW9.2	%SW9.3	%SW9.4	%SW9.5

NOTE : par défaut, les phases de lecture d'entrée et d'écriture de sortie sont actives (bits des mots système %SW8 et %SW9 réglés sur 0).

Sur Quantum, les entrées/sorties qui sont distribuées via le bus DIO ne sont pas affectées par les mots %SW8 et %SW9.

Affectation des voies d'entrées/sorties aux tâches maître, rapide et auxiliaires

Généralités

Chaque tâche assure l'écriture et la lecture des entrées/sorties qui lui ont été affectées.

L'association d'une voie, d'un groupe de voies ou d'un module d'entrées/sorties à une tâche est définie dans l'écran de configuration du module correspondant.

La tâche associée par défaut est la tâche MAST.

Lecture des entrées et écriture des sorties sur Premium

Toutes les voies d'entrées/sorties des modules en racks peuvent être associées à une tâche (MAST, FAST ou AUX 0..3).

Cas des entrées\sorties locale et distantes (bus X):

A chaque cycle de la tâche, les entrées sont lues en début de tâche et les sorties sont écrites en fin de tâche.

Cas des entrées\sorties distantes sur bus Fipio:

En mode asservi, le rafraîchissement des entrées/sorties est corrélé avec la période de la tâche. Le système garantit la mise à jour des entrées/sorties en une seule période. Seules les entrées/sorties associées à cette tâche sont rafraîchies.

Dans ce mode, la période de tâche automate (MAST, FAST ou AUX) doit être supérieure ou égale au temps de cycle réseau.

En mode libre, aucune contrainte n'est imposée sur la période de la tâche. La période de tâche automate (MAST, FAST ou AUX) peut être inférieure au temps de cycle réseau.Dans ce cas, la tâche peut être exécutée sans une mise à jour des entrées/sorties. La sélection de ce mode offre la possibilité d'avoir des temps de tâche les plus faibles possibles dans le cas d'application où la rapidité est critique.

Lecture des entrées et écriture des sorties sur Quantum

Cas des entrées\sorties locales :

Chaque module d'entrées/sorties ou groupe de modules peut être associé à une et une seule tâche (MAST, FAST ou AUX 0..3).

Cas des entrées\sorties décentralisées :

Les stations d'entrées/sorties distantes peuvent être associés uniquement à la tâche maître (MAST). L'affectation s'effectue au niveau des sections , avec 1 station d'entrées distantes et 1 station de sorties distantes par section.

Cas des entrées\sorties distribuées :

Les stations d'entrées/sorties distribuées peuvent être associées uniquement à la tâche maître (MAST).

Les entrées sont lues en début de tâche maître et les sorties sont écrites en fin de tâche maître.

Exemple sur Premium

La modularité des modules TOR Premium étant de 8 voies successives (voies 0 à 7, voies 8 à 15, ...), les entrées/sorties peuvent être affectées par groupes de 8 voies, indifféremment à la tâche MAST, AUXi ou FAST.

Exemple : il est possible d'affecter les voies d'un module 28 entrées/sorties de la manière suivante :

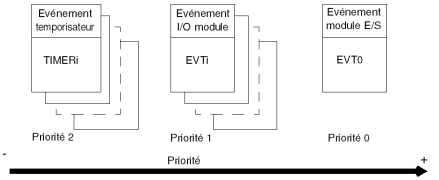
- entrées 0 à 7 affectées à la tâche MAST,
- entrées 8 à 15 affectées à la tâche FAST,
- sorties 0 à 7 affectées à la tâche MAST,
- sorties 8 à 15 affectées à la tâche AUX0.

Gestions des traitements événementiels

Généralités

Les traitements événementiels sont prioritaires sur les tâches.

L'illustration suivante décrit les 3 niveaux de priorité définis:



Gestion des priorités

- Le traitement événementiel EVT0 est le traitement le plus prioritaire. Il peut lui-même interrompre les autres traitements événementiels.
- Les traitements événementiels EVTi déclenchés par des modules d'entrées/sorties, (priorité 1) sont prioritaires sur les traitements événementiels TIMERi déclenchés par temporisateurs (priorité 2).
- **Sur les automates Modicon M340, Premium et Atrium :** les types de traitements événementiels avec un niveau de priorité 1 sont stockés et traités dans l'ordre.
- **Sur l'automate Quantum :** la priorité des types de traitement de priorité 1 est déterminée :
  - par la position du module d'entrées/sorties dans le rack,
  - par la position de la voie dans le module.

  Le module de numéro de position le plus faible a la priorité la plus grande.
- Les traitements événementiels déclenchés par temporisateur ont la priorité 2. La priorité de traitement est déterminée par le numéro de temporisateur le plus faible.

Contrôle

Le programme d'application peut globalement valider ou inhiber les différents types de traitements événementiels au travers du bit système %S38. Si un ou plusieurs événements interviennent pendant qu'ils sont inhibés, les traitements associés sont perdus.

Deux fonctions élémentaires du langage, `MASKEVT()` et `UNMASKEVT()`, utilisées dans le programme application, permettent également de masquer ou démasquer les traitements événementiels.

Si un ou plusieurs événements interviennent pendant qu'ils sont masqués, ils sont mémorisés par le système et les traitements associés ne seront effectués qu'après démasquage.

Exécution des traitements événementiels de type TIMER

Description

Les traitements événementiels de type TIMER sont des traitements déclenchés par la fonction ITCNTRL *(voir Unity Pro, Système, Bibliothèque de blocs)*.

Cette fonction de temporisation active périodiquement le traitement événementiel chaque fois que la valeur de présélection est atteinte.

Paramètres

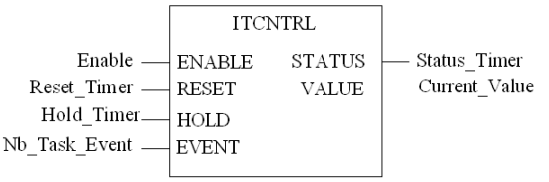
Les paramètres suivants sont sélectionnés au niveau des propriétés du traitement événementiel.

Paramètre	Valeur	Valeur par défaut	Rôle
Base de temps	1 ms, 10 ms, 100 ms, 1 s	10 ms	Base de temps du temporisateur. Remarque : la base de temps de 1 ms est à utiliser avec précaution, risque de débordement si la fréquence de déclenchement des traitements est trop importante.
Présélection	1 à 1 023	10	Valeur de présélection du temporisateur. La temporisation élaborée vaut : Présélection x Base de temps.
Phase	0 à 1 023	0	Valeur de décalage temporel entre la transition STOP/RUN de l'automate et le premier redémarrage à 0 du temporisateur. La valeur temporelle est égale : Phase x Base de temps.

NOTE : la phase doit être inférieure à la valeur prédéfinie dans l'événement de type TIMER.

Fonction ITCNTRL

Représentation en FBD :



Le tableau suivant décrit les paramètres d'entrée :

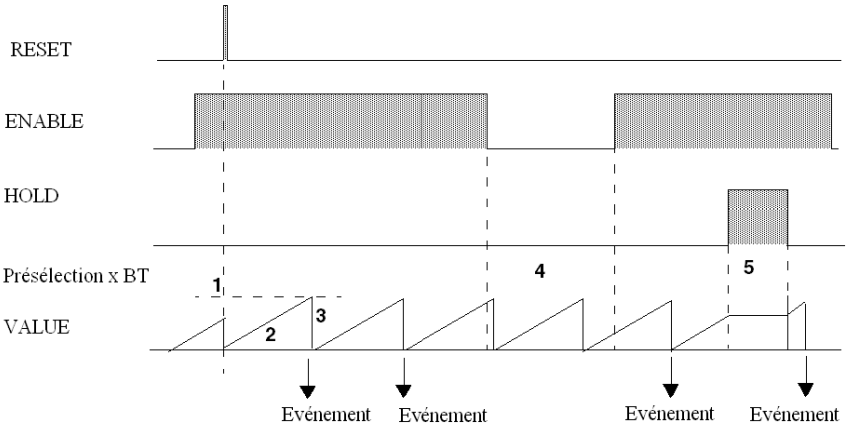
Paramètre	Type	Commentaire
Enable	BOOL	Entrée de validation
Reset_Timer	BOOL	Sur état 1 réinitialise le temporisateur.
Hold_Timer	BOOL	Sur état 1 "gèle" l'incrémentation du temporisateur.
Nb_Task_Event	BYTE	Octet d'entrée qui détermine le numéro du traitement événementiel à déclencher.

Le tableau suivant décrit les paramètres de sortie :

Paramètre	Type	Commentaire
Status_Timer	WORD	Mot d'état.
Current_Value	TIME	Valeur courante du temporisateur.

Chronogramme du fonctionnement normal

Chronogramme.



Marche normale

Le tableau suivant décrit le principe de déclenchement des traitements événementiel de type TIMER (voir chronogramme ci-dessus).

Phase	Description
1	Lors d'un front montant sur l'entrée RESET, le temporisateur est remis à 0.
2	La valeur courante VALUE du temporisateur croît de 0 vers la valeur de présélection d'une unité à chaque impulsion de la base de temps.
3	Un événement est émis dès que la valeur courante a atteint la valeur de présélection, le temporisateur est remis à 0, puis est de nouveau activé. Le traitement événementiel associé est déclenché, si l'événement n'est pas masqué. Il peut être différé si un traitement événementiel de priorité supérieure ou identique est cours d'exécution.
4	Lorsque l'entrée ENABLE est réglée sur 0, les événements ne sont plus émis. Les traitements événementiels de type TIMER ne sont plus déclenchés.
5	Quand l'entrée HOLD est réglée sur 1, le temporisateur est figé, la valeur courante n'évolue plus, tant que cette entrée ne repasse pas à 0.

Synchronisation de traitement événementiel

Le paramètre Phase permet de déclencher des traitements événementiels de type TIMER différents à intervalle de temps constant.

Ce paramètre définit un décalage temporel avec une origine de temps absolu, qui est le dernier passage de STOP en RUN de l'automate.

Condition de fonctionnement :

- Les traitements événementiels doivent avoir les mêmes valeurs de base de temps et de présélection.
- Les entrées RESET et HOLD ne doivent pas être positionnées à 1.

Exemple : 2 traitements événementiels Timer1 et Timer2 à exécuter à 70 ms d'intervalle.

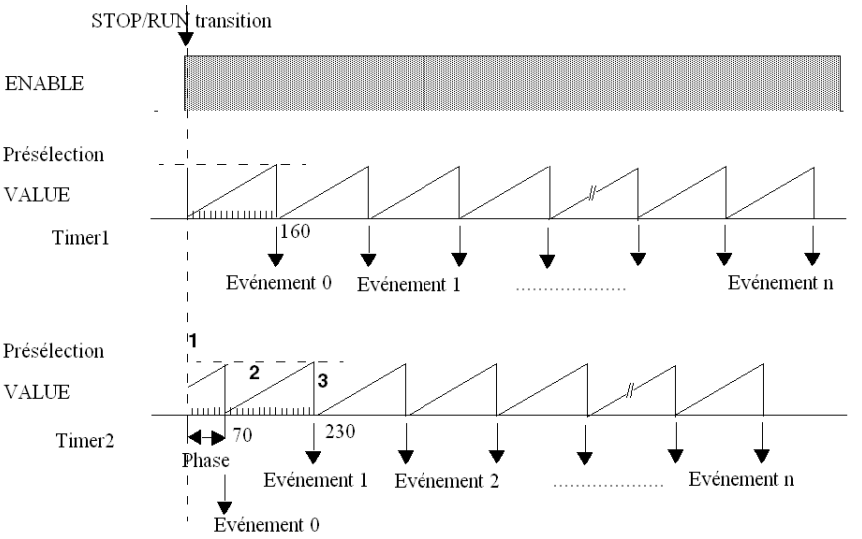
Le premier traitement Timer1 pourra être défini avec une phase égale à 0 et le second Timer2 avec une phase de 70 ms (phase de 7 et base de temps de 10 ms).

Tout événement déclenché par le temporisateur associé au traitement Timer1

sera suivi 70 ms après d'un événement issu temporisateur associé au traitement Timer2

Chronogramme : Transition STOP/RUN

Chronogramme de l'exemple décrit ci-dessus avec une même valeur de présélection de 16 (160 ms) pour Timer1 et Timer2.



Fonctionnement après un STOP/RUN pour l'automate

Le tableau suivant décrit le fonctionnement après un passage de STOP en RUN de l'automate (voir chronogramme ci-dessus) :

Phase	Description
1	Sur une transition STOP RUN de l'automate la temporisation se déclenche de façon à ce que la valeur de présélection soit atteinte au bout d'un temps égal à la Phase x base de temps, le premier événement est alors émis.
2	La valeur courante `VALUE` du temporisateur croît de 0 vers la valeur de présélection d'une unité à chaque impulsion de la base de temps.
3	Un événement est émis dès que la valeur courante a atteint la valeur de présélection, le temporisateur est remis à 0, puis est de nouveau activé. Le traitement événementiel associé est déclenché, si l'événement n'est pas masqué. Il peut être différé si un traitement événementiel de priorité supérieure ou identique est cours d'exécution.

Echanges d'entrées/de sorties dans les traitements événementiels

Généralités

Il est possible d'utiliser à chaque traitement événementiel des voies d'entrées/sorties autres que celle relative à l'événement.

Comme pour les tâches, les échanges sont alors réalisés implicitement par le système avant (%I) et après (%Q) le traitement applicatif.

Fonctionnement

Le tableau suivant décrit les échanges et les traitements réalisés.

Phase	Description
1	L'apparition d'un événement déroute le programme application vers le traitement qui est associé à la voie d'entrée/sortie qui a provoqué l'événement.
2	Toutes les entrées associées au traitement événementiel sont acquises automatiquement.
3	Le traitement événementiel est exécuté. Il doit être le plus court possible.
4	Toutes les sorties associées au traitement événementiel sont mises à jour.

Cas des Premium/Atrium

Les entrées acquises et les sorties mises à jour sont :

- les entrées associées à la voie qui a provoqué l'événement,
- les entrées et les sorties utilisées dans le traitement événementiel.

NOTE : Ces échanges peuvent être relatifs :

- à une voie (exemple module de comptage) ou
- à un groupe de voies (module TOR). Dans cas , si le traitement modifie par exemple les sorties 2 et 3 d'un module TOR, c'est l'image des sorties 0 à 7 qui sera transférée vers le module.

Cas du Quantum

Les entrées acquises et les sorties mises à jour sont sélectionnées en configuration. Seules des entrées/sorties locales peuvent être choisies.

Règle de programmation

Les entrées échangées (et le groupe de voies associées) lors de l'exécution du traitement événementiel sont remis à jour (perte des valeurs historiques, donc des fronts). Il faut donc éviter de tester des fronts sur ces entrées dans les tâches maître (MAST), rapide (FAST) ou auxiliaires (AUXi).

Programmation du traitement événementiel

Marche à suivre

Le tableau ci-dessous présente les étapes principales pour la programmation du traitement événementiel.

Etape	Action
1	**Phase de configuration** (pour les événements déclenchés par des modules d'entrées/sorties) En mode local, à partir de l'éditeur de configuration, sélectionnez **Traitement événementiel** (EVT) et le numéro du traitement événementiel de la voie du module d'entrées/sorties concerné.
2	**Phase de démasquage** La tâche qui peut être interrompue doit en particulier : ● activer le traitement des événements au niveau du système : définir le bit %S38 sur 1 (valeur par défaut) ; ● démasquer les événements à l'aide de l'instruction `UNMASKEVT` (active par défaut) ; ● démasquer les événements concernés au niveau de la voie (pour les événements déclenchés par des modules d'entrées/sorties) en définissant sur 1 les objets à langage implicite du module d'entrées/sorties pour le démasquage des événements (les événements sont masqués par défaut) ; ● vérifier que la pile d'événements au niveau du système n'est pas saturée (le bit %S39 doit être défini sur 0).
3	**Phase de création du programme d'événement** Le programme doit : ● déterminer l'origine du ou des événements à partir du mot d'état de l'événement associé au module d'entrées/sorties si le module peut générer plusieurs événements ; ● effectuer le traitement réflexe associé à l'événement (ce procédé doit être le plus court possible) ; ● écrire les sorties réflexes concernées. **Remarque** : Le mot d'état de l'événement est automatiquement remis à zéro.

Illustration du démasquage d'événements

Ce schéma présente le démasquage d'événements dans la tâche MAST.

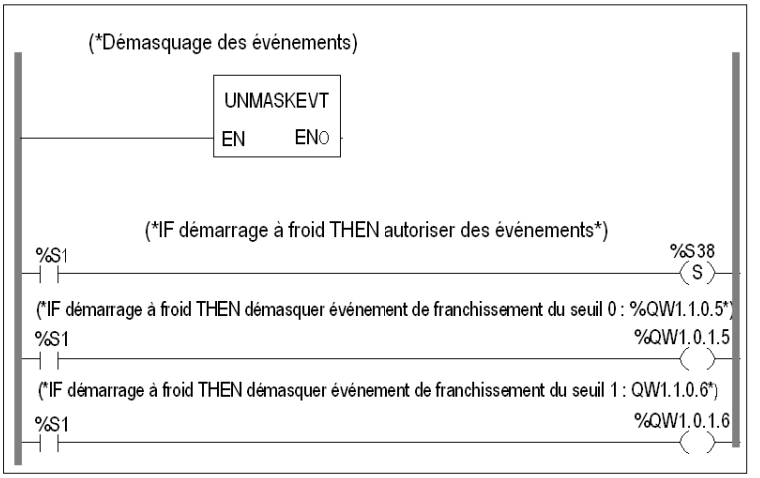
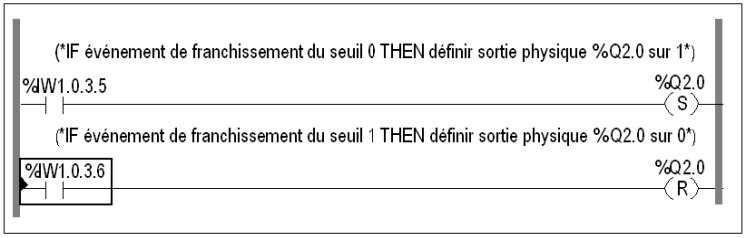


Illustration du contenu du traitement événementiel

Ce schéma présente le contenu potentiel du traitement événementiel (test et action du bit).



Structure mémoire application

4

Objet de ce chapitre

Ce chapitre décrit la structure mémoire application des automates Premium, Atrium et Quantum.

Contenu de ce chapitre

Ce chapitre contient les sous-chapitres suivants :

4.1 Structure de mémoire pour les automates Premium, Atrium et Modicon M340

Objet de cette section

Cette section décrit la structure de la mémoire des automates Premium, Atrium et Modicon M340 et en présente les différentes zones.

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Structure de mémoire des automates Modicon M340

Présentation

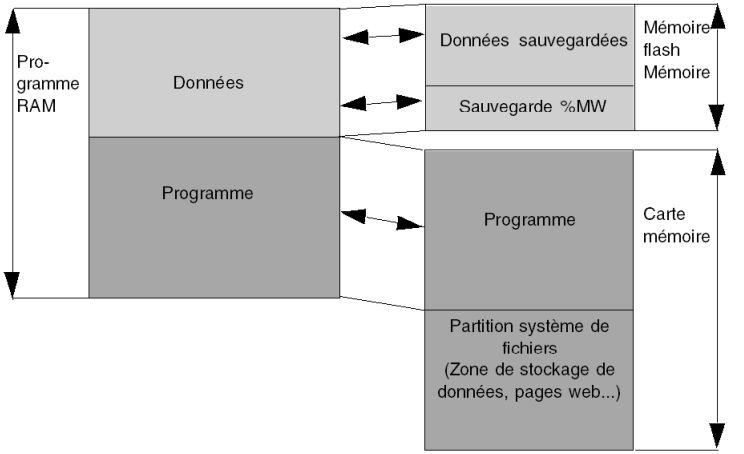
La mémoire de l'automate prend en charge :

- **les données affectées de l'application,**
- **les données non affectées de l'application,**
- **le programme** : les descripteurs et le code exécutable des tâches, les mots constants, les valeurs initiales et la configuration des entrées/sorties.

Structure

Les données et le programme sont pris en charge par la RAM interne du module processeur.

Le schéma ci-après décrit la structure de la mémoire.



Sauvegarde du programme

Si la carte mémoire est présente, fonctionne correctement et n'est pas protégée en écriture, le programme est enregistré sur la carte mémoire :

- automatiquement, après :
  - un chargement,
  - une modification en mode connecté,
  - un front montant du bit système `%S66` dans le programme du projet.
- manuellement :
  - à l'aide de la commande **Automate →Sauvegarde du projet →Enregistrer la sauvegarde**,
  - dans une table d'animation en définissant le bit système `%S66`.

⚠ AVERTISSEMENT

PERTE DE DONNEES - APPLICATION NON ENREGISTREE

L'interruption d'une procédure d'enregistrement d'une application par une extraction intempestive ou brutale de la carte mémoire peut conduire à une perte de l'application enregistrée. Le bit %S65 *(voir page 160)* permet de gérer une extraction correcte (voir l'aide sur le bit %65 dans le chapitre sur les bits système).

Le non-respect de ces instructions peut provoquer la mort, des blessures graves ou des dommages matériels.

La carte mémoire utilise la technologie Flash. Par conséquent, aucune pile n'est nécessaire.

Restauration du programme

Si la carte mémoire est présente et fonctionne correctement, le programme est copié de la carte mémoire de l'automate vers la mémoire interne :

- Automatiquement après :
  - un redémarrage.
- manuellement, à l'aide de la commande Unity Pro **Automate →Sauvegarde du projet →Restituer la sauvegarde**.

NOTE : lorsque vous insérez la carte mémoire en mode Run ou Stop, vous devez procéder à un redémarrage pour restaurer le projet sur l'automate.

Données enregistrées

Les données affectées et non affectées et le tampon de diagnostic sont enregistrés automatiquement dans la mémoire Flash interne lors de la mise hors tension. Ils sont restaurés lors d'un démarrage à chaud.

Save_Param

La fonction `SAVE_PARAM` permet de régler les paramètres actuels et initiaux dans la mémoire RAM interne (comme pour les autres automates). Dans ce cas, le contenu de la RAM interne et de la carte mémoire est différent (%S96 = 0 et le voyant CARDERR est allumé). Lors d'un démarrage à froid (après restauration de l'application), les paramètres en cours sont remplacés par les valeurs initiales définies les plus récentes, à condition qu'un enregistrement sur une carte mémoire ait été exécuté auparavant (fonction Enregistrer la sauvegarde ou front montant %S66).

Enregistrement de la valeur courante

Sur un front montant %S94, les valeurs courantes remplacent les valeurs initiales dans la mémoire interne. Le contenu de la RAM interne et de la carte mémoire est différent (%S96 = 0 et le voyant CARDERR est allumé). Lors d'un démarrage à froid, les valeurs en cours sont remplacées par les valeurs initiales les plus récentes, à condition qu'un enregistrement sur carte mémoire ait été exécuté auparavant (fonction Enregistrer la sauvegarde ou front montant %S66).

Suppression de fichiers

Il existe deux moyens de supprimer tous les fichiers de la carte mémoire :

- formatage de la carte mémoire (suppression de tous les fichiers de la partition des fichiers système),
- suppression du contenu du répertoire \DataStorage\ (suppression des fichiers ajoutés par l'utilisateur uniquement).

Les deux opérations sont effectuées à l'aide du mot système %SW93 *(voir page 186)*.

Le mot système `%SW93` peut uniquement être utilisé après le chargement d'une application par défaut dans l'automate.

⚠ ATTENTION

CARTE MEMOIRE NON OPERATIONNELLE

Ne formatez pas la carte mémoire à l'aide d'un outil autre qu'un outil Schneider. La carte mémoire a besoin d'une structure pour contenir le programme et les données. Effectuer un formatage avec un autre outil détruit cette structure.

Le non-respect de ces instructions peut provoquer des blessures ou des dommages matériels.

Sauvegarde %MW

Les valeurs de %MWi peuvent être enregistrées dans la mémoire Flash interne avec %SW96 *(voir page 186)*. Ces valeurs seront restaurées lors du redémarrage à froid, y compris du téléchargement de l'application, si l'option **Initialiser %MWi au démarrage à froid** n'est pas cochée dans l'écran Configuration du processeur.

Dans le cas des mots `%MW`, les valeurs de ces mots peuvent être enregistrées et restituées sur redémarrage à froid ou chargement si l'option de réinitialisation des `%MW` sur redémarrage à froid dans l'écran Configuration du processeur n'est pas cochée. Avec le mot `%SW96`, la gestion des mots internes `%MW` de l'action mémoire (enregistrement, suppression) et des informations sur les mots internes `%MW` des états des actions est possible.

Spécificités de la carte mémoire

Deux types de carte mémoire sont disponibles :

- **application** : ces cartes contiennent le programme d'application et les pages Web,
- **application + stockage de fichiers** : ces cartes mémoire contiennent le programme d'application, les fichiers de données des EFB de gestion des fichiers de carte mémoire et les pages Web.

Structure de mémoire des automates Premium et Atrium

Généralités

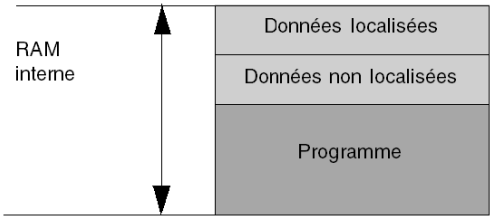
La mémoire de l'automate prend en charge :

- **les données localisées** de l'application,
- **les données non localisées** de l'application,
- **le programme** : descripteurs de tâche, code exécutable, mots constants, valeurs initiales et configuration des entrées/sorties.

Structure sans carte mémoire d'extension

Les données et programme sont supportés par la mémoire RAM interne au module processeur.

Le schéma suivant décrit la structure de la mémoire.

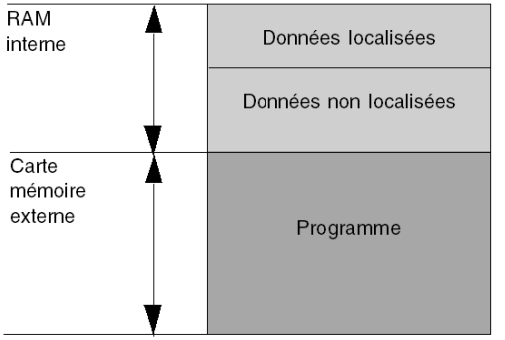


Structure avec carte mémoire d'extension

Les données sont supportées par la mémoire RAM interne au module processeur.

Le programme est supportés par la carte mémoire d'extension.

Le schéma suivant décrit la structure de la mémoire.



Sauvegarde de la mémoire

La mémoire RAM interne est secourue par la pile cadmium-nickel supportée par le module processeur.

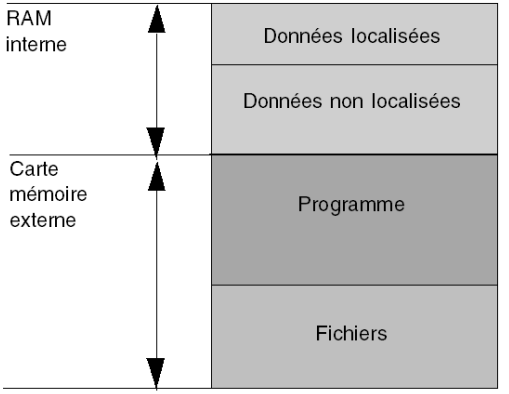
Les cartes mémoires RAM sont secourues par pile cadmium-nickel.

Spécificités des cartes mémoires

Trois types de cartes mémoires sont proposées :

- **application** : ces cartes contiennent le programme de l'application. Elles permettent d'utiliser la technologie RAM ou Flash EPROM
- **application + stockage de fichiers :** ces cartes contiennent en plus du programme, une zone permettant d'archiver/restituer des données par programme. Elles sont proposées en technologie RAM ou Flash Eprom
- **stockage de fichiers :** ces cartes permettent d'archiver/restituer des données par programme. Ces cartes sont en technologie SRAM.

Le schéma suivant décrit la structure mémoire avec carte de type application et stockage de fichiers.



NOTE : Dans le cas des processeurs possédant 2 emplacements pour carte mémoire, l'emplacement inférieur est réservé à la fonction de stockage de fichiers.

Description détaillée des zones mémoires

Données utilisateur

Cette zone contient les données localisées et les données non localisées de l'application.

- données localisées :
  - données booléennes %M, %S et numériques %MW,%SW,
  - données associées aux modules %I, %Q, %IW, %QW,%KW....
- données non localisées :
  - données booléennes et numériques (instances)
  - Instances de EFB et DFB

Programme utilisateur et constantes

Cette zone contient les codes exécutables et les constantes de l'application.

- codes exécutables :
  - code du programme
  - code associé aux EF, EFB et à la gestion des modules d'E/S
  - code associé aux DFB
- constantes :
  - mots constants KW
  - constantes associées aux entrées/sorties
  - valeurs initiales des données

Cette zone contient aussi les informations nécessaire au déchargement d'application : codes graphiques, symboles...

Autres informations

D'autres informations liées à la configuration et à la structure de l'application sont aussi stockées en mémoire (en zone données ou programme suivant le type d'information).

- Configuration: autres données liées à la configuration (configuration matérielle, configuration logicielle).
- Système: données utilisées par le système d'exploitation (pile des tâches,...).
- Diagnostic: informations liées au diagnostic du procédé ou du système, buffer de diagnostic.

4.2 Structure de la mémoire des automates Quantum

Objet de cette section

Ce chapitre décrit la structure de la mémoire des automates Quantum et en présente les différentes zones.

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Structure de la mémoire des automates Quantum

Généralités

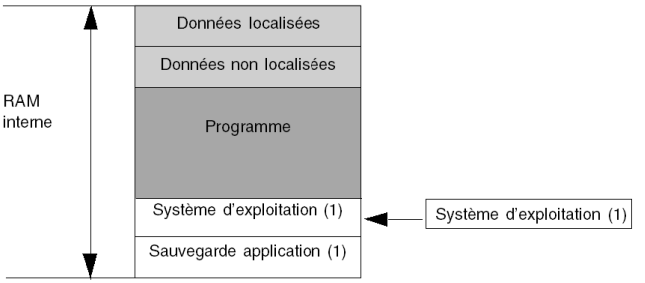
La mémoire de l'automate prend en charge :

- **les données localisées de l'application** (RAM d'état),
- **les données non localisées** de l'application,
- **le programme** : les descripteurs et le code exécutable des tâches, les valeurs initiales et la configuration des entrées/sorties.

Structure sans carte mémoire d'extension

Les données et programme sont supportés par la mémoire RAM interne au module processeur.

Le schéma suivant décrit la structure de la mémoire.



(1) uniquement sur processeurs 140 CPU 31••/43••/53••

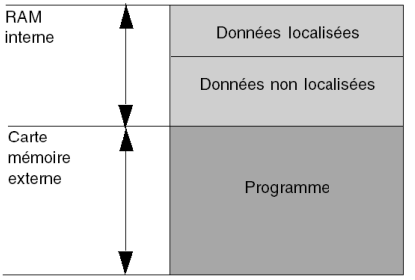
Structure avec carte mémoire d'extension

Les processeurs de l'automate Quantum 140 CPU 6••• peuvent être stockés avec une carte mémoire d'extension.

Les données sont supportées par la mémoire RAM interne au module processeur.

Le programme est supporté par la carte mémoire d'extension.

Le schéma suivant décrit la structure de la mémoire.



Sauvegarde de la mémoire

La mémoire RAM interne est secourue par la pile cadmium-nickel supportée par le module processeur.

Les cartes mémoires RAM sont secourues par pile cadmium-nickel.

Démarrage avec l'application enregistrée de la mémoire de sauvegarde

Le tableau ci-dessous décrit les différents résultats obtenus en fonction de l'état de l'automate et du switch Mem de l'automate *(voir Manuel de référence du matériel, Matériel, avec Unity Pro)*, et indique si la case à cocher "Auto RUN" est sélectionnée.

Etat de l'automate	**Switch Mem de l'automate[1]**	**Auto RUN dans appl[2]**	**Résultats**
NONCONF	Démarrage ou Désactivé	Désactivé	**Démarrage à froid**, l'application est chargée depuis la mémoire de sauvegarde vers la mémoire RAM de l'automate. L'automate reste sur STOP.
NONCONF	Démarrage ou Désactivé	Allumé	**Démarrage à froid**, l'application est chargée depuis la mémoire de sauvegarde vers la mémoire RAM de l'automate. L'automate reste sur RUN.
NONCONF	Mem Prt ou Stop	Non applicable	Aucune application chargée. L'automate démarre en mode NONCONF.
Configuré	Démarrage ou Désactivé	Désactivé	**Démarrage à froid**, l'application est chargée depuis la mémoire de sauvegarde vers la mémoire RAM de l'automate. L'automate reste sur STOP.
Configuré	Démarrage ou Désactivé	Allumé	**Démarrage à froid**, l'application est chargée depuis la mémoire de sauvegarde vers la mémoire RAM de l'automate. L'automate reste sur RUN.
Configuré	Mem Prt ou Stop	Ignoré	**Démarrage à chaud**, aucune application chargée. Mode précédent conservé à la mise sous tension.

1 Démarrage et Stop sont valides pour les modèles 434 et 534 uniquement et Désactivé est valide pour le modèle 311 uniquement. Mem Prt est valide pour tous les modèles.

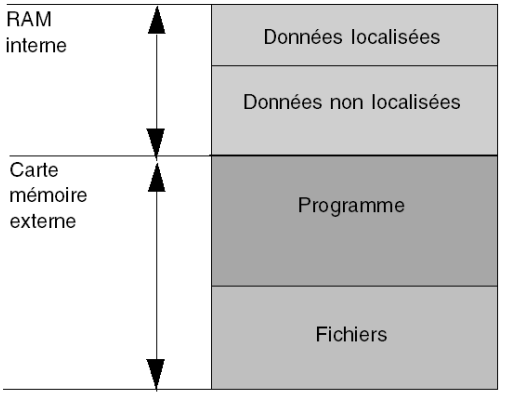
2 Le RUN automatique de l'application correspond à l'application chargée.

Spécificités des cartes mémoires

Trois types de cartes mémoires sont proposées :

- **application** : ces cartes contiennent le programme de l'application. Elles sont proposés en technologie RAM ou Flash Eprom
- **application + stockage de fichiers :** ces cartes contiennent en plus du programme, une zone permettant d'archiver/restituer des données par programme. Elles sont proposées en technologie RAM ou Flash Eprom
- **stockage de fichiers :** ces cartes permettent d'archiver/restituer des données par programme. Ces cartes sont en technologie SRAM.

Le schéma suivant décrit la structure mémoire avec carte de type application et stockage de fichiers.



NOTE : Dans le cas des processeurs possédant 2 emplacements pour carte mémoire, l'emplacement inférieur est réservé à la fonction de stockage de fichiers.

Description détaillée des zones mémoires

Données non localisées

Cette zone contient les données non localisées :

- données booléennes et numériques
- EFB et DFB

Données localisées

Cette zone contient les données localisées (State Ram).

Adresse	Repère des objets	Utilisation des données
0xxxxx	%Qr.m.c.d,%Mi	bits de module de sorties et bits internes.
1xxxxx	%Ir.m.c.d, %Ii	bits de modules d'entrées.
3xxxxx	%IWr.m.c.d, %IWi	mots registre d'entrée des modules d'entrées/sorties.
4xxxxx	%QWr.m.c.d, %MWi	mots de sortie des modules d'entrées/sorties et mots internes.

Programme utilisateur

Cette zone contient les codes exécutables de l'application.

- code du programme
- code associé aux EF, EFB et à la gestion des modules d'E/S
- code associé aux DFB
- valeurs initiales des variables

Cette zone contient aussi les informations nécessaire au déchargement d'application : codes graphiques, symboles...

Système d'exploitation

Dans le cas des processeurs 140 CPU 31••/41••/51••, cette zone contient le système d'exploitation pour le traitement de l'application. Ce système d'exploitation est transféré d'une mémoire interne EPROM vers la mémoire interne RAM lors de la mise sous tension.

Sauvegarde d'application

Une zone mémoire Flash EPROM de 1435K8, disponible sur les processeurs 140 CPU 31••/41••/51••, permet la sauvegarde du programme et des valeurs initiales des variables.

L'application sauvegardée dans cette zone est transférée automatiquement dans la RAM interne à la mise sous tension du processeur automate (si le switch MEM de l'automate est en position off en face avant du processeur automate).

Autres informations

D'autres informations liées à la configuration et à la structure de l'application sont aussi stockées en mémoire (en zone données ou programme suivant le type d'information).

- Configuration: autres données liées à la configuration (configuration matérielle, configuration logicielle).
- Système: données utilisées par le système d'exploitation (pile des tâches,...).
- Diagnostic: informations liées au diagnostic du procédé ou du système, buffer de diagnostic.

Modes de fonctionnement

5

Objet de ce chapitre

Ce chapitre décrit les modes de fonctionnement de l'automate en cas de coupure et de reprise secteur, les incidences sur le programme application et la mise à jour des entrées/sorties.

Contenu de ce chapitre

Ce chapitre contient les sous-chapitres suivants :

5.1 Modes de fonctionnement des automates Modicon M340

Objet de cette section

Cette section décrit les modes de fonctionnement des automates Modicon M340.

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Traitement en cas de coupure de courant et restauration des automates Modicon M340

Généralités

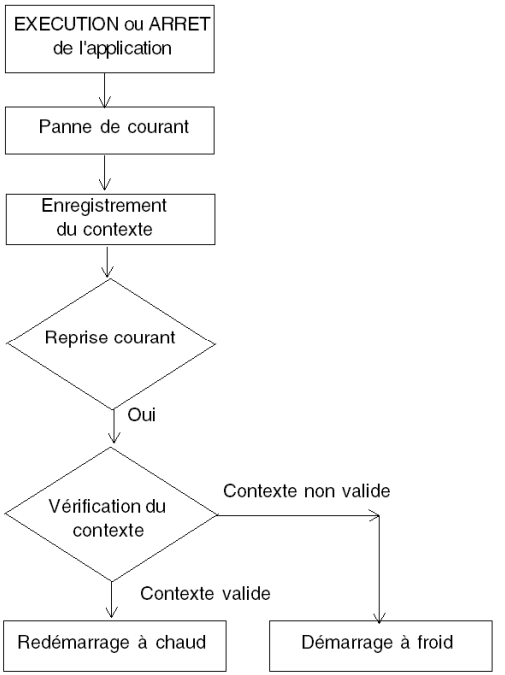
Si la durée de la coupure est inférieure au temps de filtrage de l'alimentation, il n'y a aucune incidence sur le programme qui s'exécute normalement. Dans le cas inverse, il y a interruption du programme et activation de la restauration de l'alimentation.

Temps de filtrage :

Automate	Courant alternatif	Courant continu
BMX CPS 2000 BMX CPS 3500 BMX CPS 3540T	10 ms	-
BMX CPS 2010 BMX CPS 3020	-	1 ms

Illustration

L'illustration suivante montre les phases du cycle d'alimentation.



Fonctionnement

Le tableau suivant décrit les phases du traitement des coupures de courant.

Etape	Description
1	En cas de coupure de courant, le système enregistre le contexte applicatif, les valeurs des variables d'application et l'état du système dans la mémoire Flash interne.
2	Le système configure toutes les sorties à l'état de repli (état défini par configuration).
3	Dès l'alimentation restaurée, certaines actions et vérifications sont effectuées pour vérifier si le redémarrage à chaud est disponible : ● restauration du contexte applicatif à partir de la mémoire Flash interne, ● vérification avec la carte mémoire (présence, disponibilité de l'application), ● vérification du contexte applicatif pour voir s'il est identique à celui de la carte mémoire. Si toutes les vérifications sont correctes, un redémarrage à chaud *(voir page 130)* est effectué, sinon un redémarrage à froid *(voir page 125)* a lieu.

Traitement sur départ à froid pour les automates Modicon M340

Cause d'un démarrage à froid

Le tableau suivant décrit les différentes causes possibles d'un démarrage à froid.

Causes	Caractéristiques du démarrage
Chargement d'une application	Démarrage à froid forcé en STOP
Rétablissement d'une application à partir d'une carte mémoire lorsque l'application est différente de celle dans la RAM interne	Démarrage à froid forcé en STOP ou en RUN selon définition en configuration
Rétablissement d'une application à partir d'une carte mémoire à l'aide de commandes Unity Pro **Automate →Sauvegarde du projet →....**	Démarrage à froid forcé en STOP ou en RUN selon définition en configuration
Activation du bouton RESET de l'alimentation	Démarrage à froid forcé en mode STOP ou RUN selon la définition de la configuration
Activation du bouton RESET sur l'alimentation moins de 500 ms après une mise hors tension	Démarrage à froid forcé en mode STOP ou RUN selon la définition de la configuration
Activation du bouton RESET sur l'alimentation après une erreur de processeur, sauf dans le cas d'une erreur de chien de garde	Démarrage à froid forcé en mode STOP Le démarrage en mode RUN défini dans la configuration n'est pas pris en compte
Initialisation depuis Unity Pro Forçage du bit système %S0	Démarrage en mode STOP ou RUN (conserve le mode de fonctionnement en cours), initialisation uniquement des applications
Reprise après une coupure de l'alimentation avec perte du contexte	Démarrage à froid forcé en mode STOP ou RUN selon la définition de la configuration

⚠ ATTENTION

PERTE DE DONNEES LORS DU TRANSFERT DE L'APPLICATION

le chargement ou le transfert d'une application sur l'automate implique normalement l'initialisation des variables non affectées.

Pour sauvegarder les variables locales :

- Evitez l'initialisation de `%MWi` en désélectionnant **Initialisation de %MWi au démarrage à froid** dans l' **écran de configuration** de l'UC.

Il est nécessaire d'affecter aux données une adresse topologique si le process impose de conserver les valeurs courantes de ces données lors du transfert de l'application.

Le non-respect de ces instructions peut provoquer des blessures ou des dommages matériels.

⚠ ATTENTION

PERTE DE DONNEES LORS DU TRANSFERT DE L'APPLICATION

N'appuyez pas sur le bouton RESET de l'alimentation. Sinon, %MWi est réinitialisé et les valeurs initiales sont chargées.

Le non-respect de ces instructions peut provoquer des blessures ou des dommages matériels.

⚠ ATTENTION

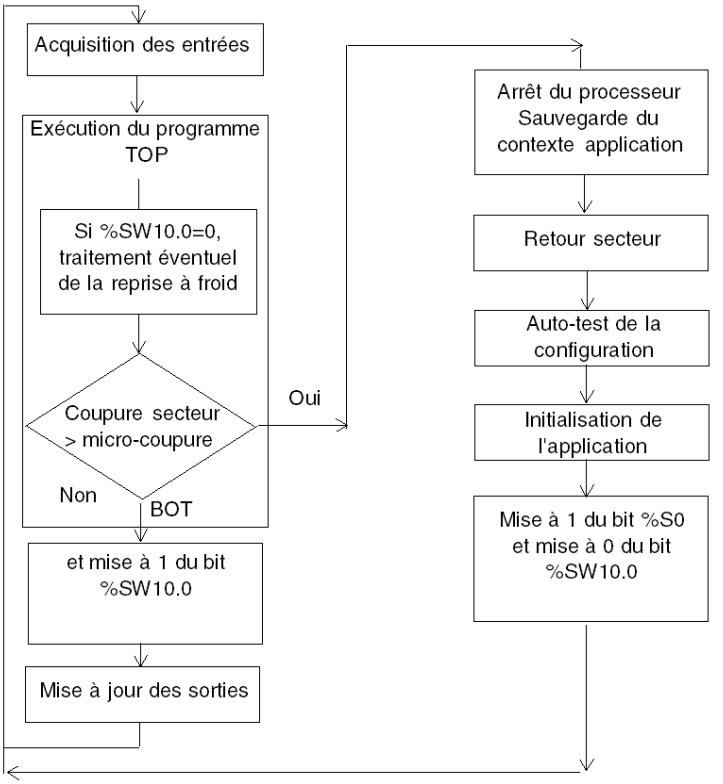
RISQUE DE PERTE D'APPLICATION

S'il n'y a pas de carte mémoire dans l'automate lors d'un redémarrage à froid, l'application est perdue.

Le non-respect de ces instructions peut provoquer des blessures ou des dommages matériels.

Illustration

Le schéma ci-après décrit le déroulement d'un redémarrage à froid.



Opération

Le tableau ci-après décrit les phases de reprise de l'exécution du programme lors d'un redémarrage à froid.

Phase	Description
1	Le démarrage est effectué en mode RUN ou STOP selon l'état du paramètre `Démarrage automatique en RUN` défini dans la configuration ou, le cas échéant, selon l'état de l'entrée RUN/STOP. L'exécution du programme reprend en début de cycle.

2	Le système effectue :
	● la désactivation des tâches, autres que la tâche maître, jusqu'à la fin du premier cycle de la tâche maître,
	● l'initialisation de données (bits, image des E/S, mots...) par les valeurs initiales définies dans l'éditeur de données (valeur réglée sur 0 si aucune valeur initiale n'est définie). Pour les mots %MW, les valeurs peuvent être récupérées sur redémarrage à froid si les deux conditions suivantes s'appliquent :
	  ● l'option **Initialiser %Mwi au redémarrage à froid** *(voir Unity Pro, Modes de marche,)* n'est pas cochée dans l'écran de configuration du processeur,
	  ● la mémoire flash interne a une sauvegarde valide (voir %SW96 *(voir page 186)*).
	  Remarque : si le nombre de mots %MW dépasse la taille de la sauvegarde (voir la structure de mémoire des automates M340 *(voir page 107)*) pendant l'opération d'enregistrement, les mots restants sont réglés sur 0.
	● l'initialisation des blocs fonction élémentaires à partir des données initiales,
	● l'initialisation des données déclarées dans les DFB : soit à 0, soit à la valeur initiale déclarée dans le type du DFB,
	● l'initialisation des bits et mots système,
	● le positionnement des graphes sur les étapes initiales,
	● l'annulation de tout forçage,
	● l'initialisation des files de messages et d'événements,
	● l'envoi des paramètres de configuration à tous les modules d'entrées/sorties TOR et métiers.

3	Pour ce premier cycle de reprise le système effectue : ● la relance de la tâche maître avec les bits %S0 (redémarrage à froid) et %S13 (premier cycle en RUN) réglés sur 1, le mot %SW10 (détection d'un redémarrage à froid lors du premier cycle d'une tâche) est réglé sur 0, ● la remise à 0 des bits %S0 et %S13 et le réglage sur 1 de chaque bit du mot %SW10 à la fin de ce premier cycle de la tâche maître, ● l'activation de la tâche rapide et des traitements événementiels à la fin du premier cycle de la tâche maître.

Traitement par programme d'un démarrage à froid

Il est conseillé de tester le bit %SW10.0 pour détecter un démarrage à froid et lancer un traitement spécifique à ce démarrage à froid.

NOTE : il est possible de tester le bit %S0 si le paramètre `Démarrage automatique en RUN` a été sélectionné. Si ce n'est pas le cas, l'automate démarre en mode STOP, le bit %S0 passe alors à 1 au premier cycle après le redémarrage, mais reste invisible pour le programme parce qu'il n'est pas exécuté.

Changements de sortie

Dès la détection d'une coupure de courant, les sorties sont mises en position de repli :

- soit elles prennent la valeur de repli,
- soit il y a maintien de la valeur en cours,

selon le choix de la configuration.

Au retour de l'alimentation, les sorties sont à zéro jusqu'à ce qu'elles soient remises à jour par la tâche.

Traitement sur départ à froid pour les automates Modicon M340

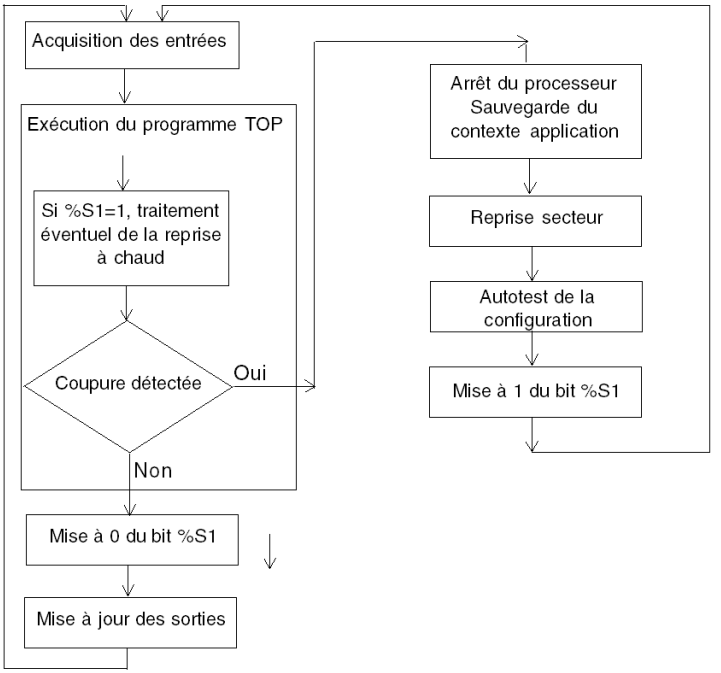
Cause d'une reprise à chaud

Une reprise à chaud peut être provoquée par un rétablissement du courant sans perte du contexte.

⚠ ATTENTION
RISQUE DE PERTE D'APPLICATION S'il n'y a pas de carte mémoire dans l'automate lors d'une reprise à chaud, l'application est perdue. **Le non-respect de ces instructions peut provoquer des blessures ou des dommages matériels.**

Illustration

Le schéma ci-après décrit le déroulement d'une reprise à chaud.



Opération

Le tableau ci-après décrit les phases de reprise de l'exécution du programme sur reprise à chaud.

Etape	Description
1	L'exécution du programme ne reprend pas à partir de l'élément où a eu lieu la coupure de courant. Le programme restant est ignoré pendant le démarrage à chaud. Chaque tâche redémarre depuis le début.
2	A la fin du cycle de reprise, le système effectue : ● la restitution de la valeur variable de l'application, ● la définition du bit %S1 sur 1, ● l'initialisation des files de messages et d'événements, ● l'envoi des paramètres de configuration à tous les modules d'entrées/sorties TOR et métiers, ● la désactivation de la tâche rapide et des traitements événementiels (jusqu'à la fin du premier cycle de la tâche maître).
3	Le système effectue un cycle de reprise dans lequel il : ● relance la tâche maître depuis le début du cycle, ● remet à l'état 0 le bit %S1 à la fin de ce premier cycle de la tâche maître, ● réactive la tâche rapide et des traitements événementiels à la fin de ce premier cycle de la tâche maître.

Traitement par programme de la reprise à chaud

En cas d'une reprise à chaud, si vous désirez un traitement particulier vis-à-vis de l'application, vous devez écrire le programme correspondant pour tester que %S1 est défini sur 1 en début de programme de la tâche maître.

Fonctions spécifiques du démarrage à chaud SFC

Le démarrage à chaud sur les automates M340 n'est pas réellement considéré comme un démarrage à chaud par l'UC. L'interpréteur SFC ne dépend pas des tâches.

SFC publie une zone de mémoire "ws_data" sur le système d'exploitation qui contient les données spécifiques à la section SFC qui doivent être enregistrées en cas de panne. Au début du traitement du diagramme, les étapes actives sont enregistrées dans "ws_data" et le traitement est marqué comme étant dans une "section critique". Une fois le traitement terminé, la section n'est plus marquée comme critique.

Si une coupure de courant se produit dans cette section critique, elle peut être décelée si cet état est actif au début (puisque le cycle est abandonné et que la tâche MAST est recommencée au début). Dans ce cas, l'espace de travail peut être incohérent et est restauré à partir des données sauvegardées.

D'autres informations provenant de SFCSTEP_STATE dans la zone des données localisées sont utilisées pour régénérer l'état de la machine.

Lorsqu'une panne de courant se produit :

- au cours du premier cycle %S1 =1, la tâche Mast est exécutée mais pas les tâches Fast et Event.

Lors du rétablissement du courant :

- le diagramme est effacé, l'enregistrement du diagnostic est annulé, les actions définies sont conservées,
- les étapes sont définies à partir de la zone sauvegardée,
- les temps d'étapes sont définis à partir de SFCSTEP_STATE,
- le temps écoulé est restauré pour les actions minutées.

NOTE : L'interpréteur SFC est indépendant, si la transition est valide, le diagramme SFC évolue tant que %S1 est vrai.

Changements de sortie

Dès la détection de la coupure de courant, les sorties sont mises en position de repli :

- soit elles prennent la valeur de repli,
- soit il y a maintien de la valeur en cours,

suivant le choix de la configuration.

Au rétablissement du courant, les sorties restent en mode sécurité (état égal à 0) jusqu'à leur mise à jour par une tâche d'exécution.

Démarrage automatique en mode RUN des automates Modicon M340

Description

Le démarrage automatique en mode RUN est une option de configuration du processeur. Cette option force l'automate à démarrer en mode RUN après un redémarrage à froid *(voir page 125)*, sauf après le chargement d'une application sur ce dernier.

Pour les modules Modicon M340, cette option n'est pas prise en compte lorsque le bouton de réinitialisation de l'alimentation est activé après une erreur liée au processeur, sauf si celle-ci concerne le chien de garde.

⚠ AVERTISSEMENT

COMPORTEMENT INATTENDU DU SYSTEME - DEBUT INATTENDU DU PROCESSUS

Les actions ci-après déclenchent le démarrage automatique en RUN :
- restitution de l'application depuis la carte mémoire,
- usage fortuit ou imprudent du bouton de réinitialisation.

Pour éviter un redémarrage non souhaité en mode RUN, utilisez :
- l'entrée RUN/STOP sur les automates Modicon M340

Le non-respect de ces instructions peut provoquer la mort, des blessures graves ou des dommages matériels.

5.2 Mode de fonctionnement des automates Premium et Quantum

Objet de cette section

Cette section décrit les modes de fonctionnement des automates Premium et Quantum.

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Traitement en cas de coupure et de reprise secteur des automates Premium/Quantum

Généralités

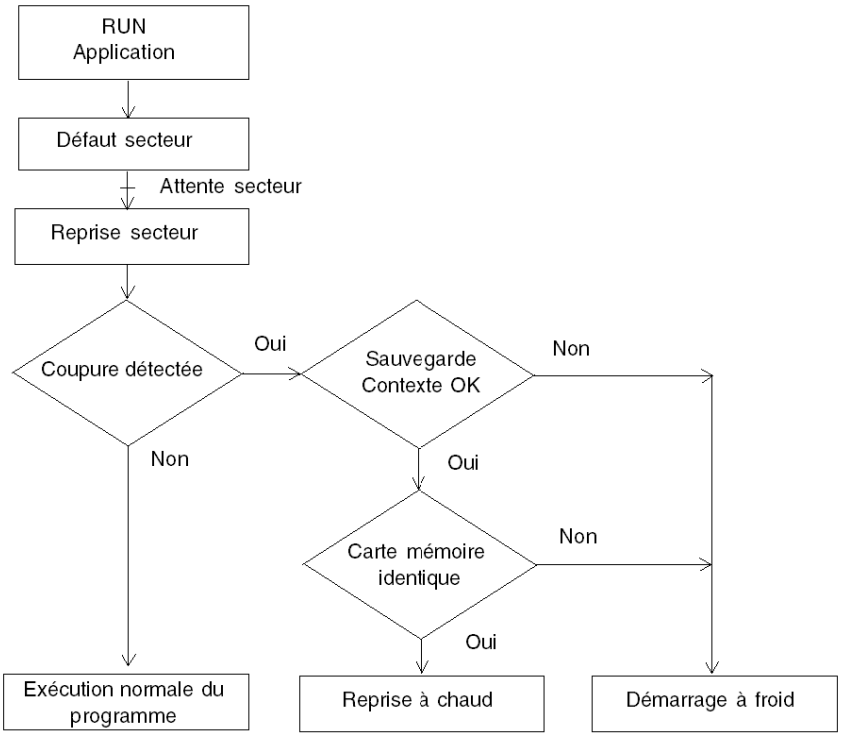
Si la durée de la coupure est inférieure au temps de filtrage de l'alimentation, celle-ci n'est pas vue par le programme qui s'exécute normalement. Dans le cas inverse, il y a interruption du programme et traitement de la reprise secteur.

Temps de filtrage :

automate	Alimentation alternative	Alimentation continue
Premium	10ms	1ms
Atrium	30ms	-
Quantum	10ms	1ms

Illustration

L'illustration présente les différentes reprises secteurs détectées par le système.



Fonctionnement

Le tableau ci-après décrit les phases du traitement des coupures secteur.

Phase	Description
1	Lors de la coupure secteur, le système mémorise le contexte application et l'heure de la coupure.
2	Il positionne toutes les sorties à l'état repli (état défini par configuration).
3	A la reprise secteur, le contexte sauvegardé est comparé à celui en cours; ce qui définit le type de démarrage à exécuter : ● si le contexte application a changé (perte du contexte système ou nouvelle application), l'automate effectue l'initialisation de l'application : démarrage à froid, ● si le contexte application est identique, l'automate effectue une reprise sans initialisation des données : reprise à chaud.

Coupure de l'alimentation sur un rack, autre que le rack 0

Toutes les voies de ce rack sont vues en erreur par le processeur mais les autres racks ne sont pas perturbés. Les valeurs des entrées en erreur ne sont plus rafraîchies dans la mémoire application et sont mises à 0 dans le cas d'un module d'entrée TOR à moins quelles aient été forcées auquel cas, elles sont maintenues à la valeur de forçage.

Si la durée de la coupure est inférieure au temps de filtrage, celle-ci n'est pas vue par le programme qui s'exécute normalement.

Traitement des automates Premium/Quantum lors d'un démarrage à froid

Cause d'un démarrage à froid

Le tableau suivant décrit les différentes causes possibles d'un démarrage à froid.

Causes	Caractéristiques du démarrage
Chargement d'une application	Démarrage à froid forcé en STOP
Activation du bouton RESET du processeur (Premium)	Démarrage à froid forcé en STOP ou en RUN selon définition en configuration
Activation du bouton RESET du processeur après une erreur processeur ou système (Premium).	Démarrage à froid forcé en STOP
Manipulation du préhenseur ou insertion/extraction d'une carte mémoire PCMCIA	Démarrage à froid forcé en STOP ou en RUN selon définition en configuration
Initialisation depuis Unity Pro Forçage du bit système %S0	Démarrage en STOP ou en RUN (conserve le mode de fonctionnement en cours), sans initialisation des modules d'entrées/sorties TOR et métier
Reprise après une coupure de l'alimentation avec perte du contexte	Démarrage à froid forcé en STOP ou en RUN selon définition en configuration

⚠ ATTENTION

PERTE DE DONNEES LORS DU TRANSFERT DE L'APPLICATION

le chargement ou le transfert d'une application sur l'automate implique normalement l'initialisation des variables non affectées.

Pour enregistrer les variables affectées sur les automates Premium et Quantum :

- Enregistrez et restaurez `%M` et `%MW` en cliquant sur **Automate →Transfert de données**.

Pour les automates Premium :

- Evitez l'initialisation de `%MW` en désélectionnant **Initialiser %MWi au démarrage à froid** dans l'**écran de configuration** de l'UC.

Pour les automates Quantum :

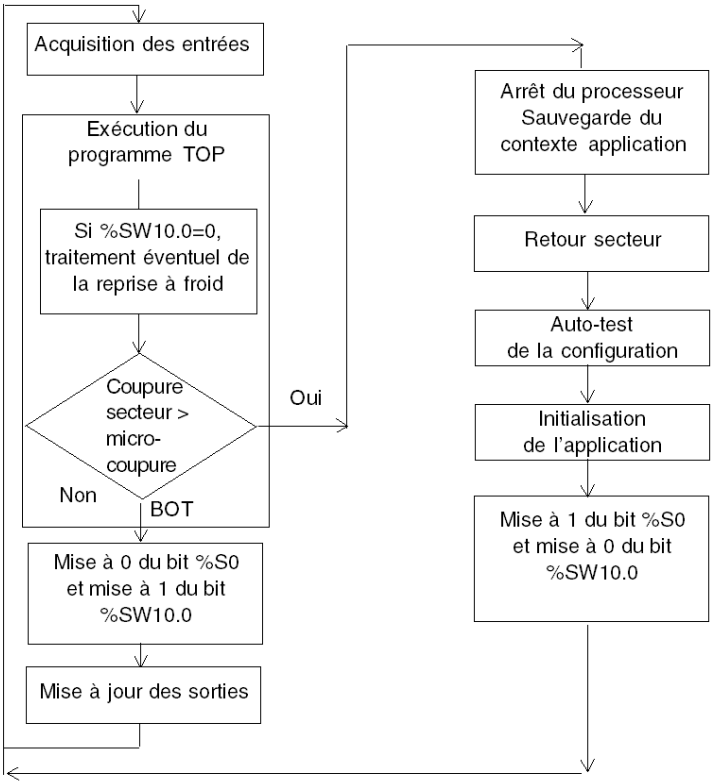
- Evitez l'initialisation de `%MW` en désélectionnant **RAZ %MWi** dans l'**écran de configuration** de l'UC.

Il est nécessaire d'affecter aux données une adresse topologique si le process impose de conserver les valeurs courantes de ces données lors du transfert de l'application.

Le non-respect de ces instructions peut provoquer des blessures ou des dommages matériels.

Illustration

Le schéma ci-après décrit le déroulement d'un redémarrage à froid.



Opération

Le tableau ci-après décrit les phases de reprise de l'exécution du programme lors d'un redémarrage à froid.

Phase	Description
1	Le démarrage est effectué en mode RUN ou STOP, selon l'état du paramètre `Démarrage automatique en RUN` défini dans la configuration ou, le cas échéant, selon l'état de l'entrée RUN/STOP. L'exécution du programme reprend en début de cycle.
2	Le système effectue : ● l'initialisation de données (bits, image des E/S, mots...) par les valeurs initiales définies dans l'éditeur de données (valeur à 0 si aucune valeur initiale n'a été définie). Dans le cas des %MW, les valeurs de ces mots peuvent être conservées au redémarrage à froid si l'option RAZ %MW au redémarrage à froid dans l'écran Configuration du processeur n'est pas cochée. ● l'initialisation des blocs fonction élémentaires à partir des données initiales, ● l'initialisation des données déclarées dans les DFB : soit à 0, soit à la valeur initiale déclarée dans le type du DFB, ● l'initialisation des bits et mots système, ● la désactivation des tâches, hormis la tâche maître, jusqu'à la fin du premier cycle de la tâche maître, ● le positionnement des graphes sur les étapes initiales, ● l'annulation des forçages, ● l'initialisation des files de messages et d'événements, ● l'envoi des paramètres de configuration à tous les modules d'entrées/de sorties TOR et métiers.
3	Pour ce premier cycle de reprise le système effectue : ● la relance de la tâche maître avec les bits %S0 (redémarrage à froid) et %S13 (premier cycle en RUN) réglés sur 1, le mot %SW10 (détection d'un redémarrage à froid lors du premier cycle d'une tâche) est réglé sur 0, ● la remise à 0 des bits %S0 et %S13 et le réglage sur 1 de chaque bit du mot %SW10 à la fin de ce premier cycle de la tâche maître, ● l'activation de la tâche rapide et des traitements événementiels à la fin du premier cycle de la tâche maître.

Traitement par programme d'un démarrage à froid

Il est conseillé de tester le bit %SW10.0 pour détecter un démarrage à froid et lancer un traitement spécifique à ce démarrage à froid.

NOTE : il est possible de tester le bit %S0 si le paramètre `Démarrage automatique RUN` a été sélectionné. Si ce n'est pas le cas, l'automate démarre en mode STOP, le bit %S0 passe alors à 1 au premier cycle après le redémarrage, mais reste invisible pour le programme parce qu'il n'est pas exécuté.

Evolution des sorties, pour les modules Premium et Atrium

Dès la détection d'une coupure de courant, les sorties sont mises en position de repli :

- soit elles prennent la valeur de repli,
- soit il y a maintien de la valeur en cours,

selon le choix de la configuration.

Au retour de l'alimentation, les sorties sont à zéro jusqu'à ce qu'elles soient remises à jour par la tâche.

Evolution des sorties, pour le Quantum

Dès la détection d'une panne d'alimentation,

- les sorties locales sont mises à zéro,
- les sorties des racks d'extension décentralisés ou distribués sont mises en position de repli.

Au retour de l'alimentation, les sorties sont à zéro jusqu'à ce qu'elles soient remises à jour par la tâche.

NOTE : le comportement des sorties forcées dans Modsoft/NxT/Concept et Unity Pro a été modifié.

Avec Modsoft/NxT/Concept, vous ne pouvez pas forcer les sorties si l'interrupteur de protection de la mémoire du processeur Quantum est en position « On ».

Avec Unity Pro, vous pouvez forcer les sorties si l'interrupteur de protection de la mémoire du processeur Quantum est en position « On ».

Avec Modsoft/NxT/Concept, les sorties forcées conservent leur état après un démarrage à froid.

Avec Unity Pro, les sorties forcées perdent leur état après un démarrage à froid.

⚠ ATTENTION

COMPORTEMENT INATTENDU DE L'APPLICATION - VARIABLES FORCEES

Vérifiez vos variables forcées et l'interrupteur de protection de la mémoire lors d'une transition entre Modsoft/NxT/Concept et Unity Pro.

Le non-respect de ces instructions peut provoquer des blessures ou des dommages matériels.

Pour les processeurs Quantum 140 CPU 31••/41••/51••

Ces processeurs sont dotés d'une mémoire Flash EPROM de 1 435 Ko, qui peut être utilisée pour enregistrer le programme et les valeurs initiales des variables.

Lors du rétablissement de l'alimentation, vous pouvez choisir le mode de marche souhaité à l'aide du commutateur PLC MEM placé en face avant du processeur. Pour plus d'informations sur le fonctionnement de ce commutateur, consultez le manuel Quantum.

- **Position « Off » :** l'application contenue dans la mémoire est transférée automatiquement dans la RAM interne à la mise sous tension du processeur automate : redémarrage à froid de l'application.
  Position « On » : l'application contenue dans la mémoire n'est pas transférée vers la RAM interne : reprise à chaud de l'application.

Traitement des automates Premium/Quantum lors d'une reprise à chaud

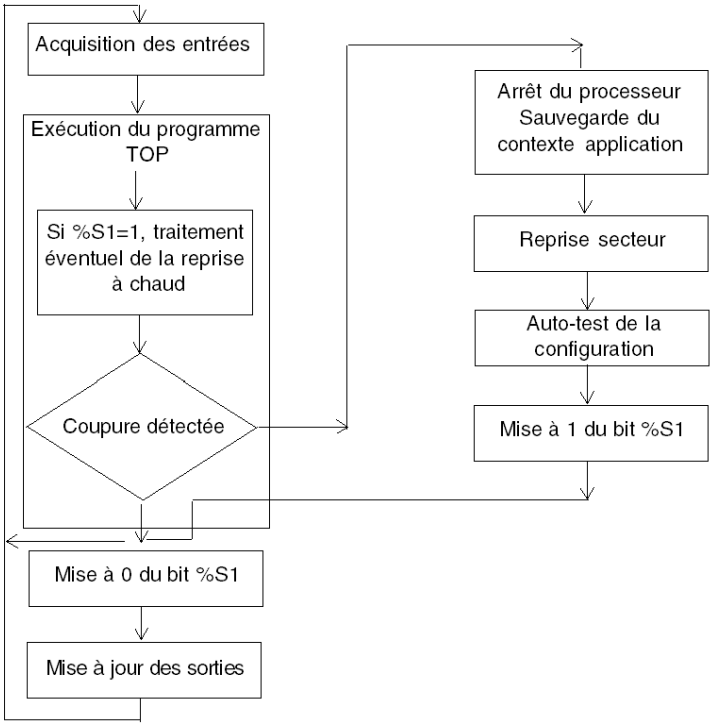
Cause d'une reprise à chaud

Une reprise à chaud peut être provoquée :

- par une reprise secteur sans perte du contexte,
- par définition sur 1 par programme du bit système %S1,
- depuis Unity Pro par bornier,
- par activation du bouton RESET du module d'alimentation du rack 0 (sur l'automate Premium).

Illustration

Le schéma ci-après décrit le déroulement d'une reprise à chaud.



Opération

Le tableau ci-après décrit les phases de reprise de l'exécution du programme sur reprise à chaud.

Etape	Description
1	L'exécution du programme reprend à partir de l'élément où a eu lieu la coupure secteur, sans mise à jour des sorties.
2	A la fin du cycle de reprise, le système effectue : ● l'initialisation des files de messages et d'événements, ● l'envoi des paramètres de configuration à tous les modules d'entrées/de sorties TOR et métiers, ● la désactivation de la tâche rapide et des traitements événementiels (jusqu'à la fin du premier cycle de la tâche maître).
3	Le système effectue un cycle de reprise dans lequel il : ● reprend en compte l'ensemble des modules d'entrées, ● relance la tâche maître avec le bit %S1 (reprise à chaud) défini sur 1, ● redéfinit le bit %S1 sur 0 à la fin de ce premier cycle de la tâche maître, ● réactive la tâche rapide, les tâches auxiliaires et les traitements événementiels à la fin de ce premier cycle de la tâche maître.

Traitement par programme de la reprise à chaud

En cas de reprise à chaud, si vous désirez un traitement particulier vis-à-vis de l'application, vous devez écrire le programme correspondant sur test de %S1 à 1 en début de programme de la tâche maître.

Sur automates Quantum, le commutateur situé en face avant du processeur permet de configurer les modes de marche. Pour plus de détails, reportez vous à la documentation Quantum.

Evolution des sorties, pour les modules Premium et Atrium

Dès la détection de la coupure de courant, les sorties sont mises en position de repli :

- elles prennent la valeur de repli ou
- il y a maintien de la valeur en cours.

suivant le choix de la configuration.

Au rétablissement du courant, les sorties sont en position de repli jusqu'à ce qu'elles soient remises à jour par la tâche.

NOTE : en cas de mise sous tension lorsque l'UC n'est pas démarrée, les sorties sont en mode sécurité (état égal à 0). Une fois l'UC démarrée, si le module n'est pas maintenu sous tension, l'état de maintien est perdu et la sortie reste à l'état 0.

Evolution des sorties, pour le Quantum

Dès la détection de la coupure secteur :
- les sorties locales sont mises à zéro,
- les sorties des racks d'extension décentralisés ou distribués sont mises en position de repli.

Au rétablissement du courant, les sorties sont en position de repli jusqu'à ce qu'elles soient remises à jour par la tâche.

Evolution des sorties, pour le rack d'extension

En cas de coupure de courant sur le rack où se trouve l'UC :
- état de repli dès la détection d'une perte d'UC,
- état de sécurité pendant la configuration des E/S,
- état calculé par l'UC après la première exécution de la tâche gérant cette sortie.

Après le rétablissement du courant, les sorties sont en position de repli jusqu'à ce qu'elles soient remises à jour par la tâche.

Démarrage automatique en mode RUN pour Premium/Quantum

Description

Le démarrage automatique en mode RUN est une option de configuration du processeur. Cette option force l'automate à démarrer en mode RUN après un redémarrage à froid *(voir page 137)*, sauf après le chargement d'une application sur ce dernier.

Pour les automates Quantum, le démarrage automatique en mode RUN dépend de la position du commutateur situé sur le panneau avant du processeur. Pour plus de détails, reportez-vous à la documentation Quantum.

⚠ AVERTISSEMENT

COMPORTEMENT INATTENDU DU SYSTEME - DEBUT INATTENDU DU PROCESSUS

Les actions ci-après déclenchent le démarrage automatique en RUN :
- l'insertion d'une carte PCMCIA à la mise sous tension de l'automate (Premium, Quantum),
- le remplacement du processeur sous tension (Premium, Quantum),
- l'usage fortuit ou imprudent du bouton de réinitialisation,
- une pile qui s'avère défectueuse en cas de coupure de courant (Premium, Quantum).

Pour éviter un redémarrage non souhaité en mode RUN :
- Il est vivement recommandé d'employer la commande RUN/STOP sur les automates Premium ou le commutateur situé à l'avant du panneau du processeur sur les automates Quantum.
- Il est **déconseillé** d'utiliser les entrées mémorisées en tant qu'entrée RUN/STOP pour l'automate.

Le non-respect de ces instructions peut provoquer la mort, des blessures graves ou des dommages matériels.

5.3 Mode HALT de l'automate

Mode HALT de l'automate

Présentation

L'automate est passé en HALT par les actions suivantes:

- utilisation de l'instruction HALT
- débordement du chien de garde
- erreur d'exécution du programme (division par zéro, débordement,...) si le bit %S78 *(voir page 160)* est positionné à 1.

Précautions

Attention ; lorsque l'automate est en HALT toutes les tâches sont stoppées *(voir Unity Pro, Modes de marche,)*. Vérifiez le comportement des E/S associées.

Objets système

6

Objet de ce chapitre

Ce chapitre décrit les bits et mots système du langage Unity Pro.

Note : les symboles, associés à chaque objet bit ou mot système, mentionnées dans les tableaux descriptifs de ces objets ne sont pas implémentés de base dans le logiciel, ils peuvent être saisis à l'aide de l'éditeur de données.

Ils sont proposés afin de rendre homogène leur appellation dans les différentes applications.

Contenu de ce chapitre

Ce chapitre contient les sous-chapitres suivants :

6.1 Bits système

Objet de cette section

Ce chapitre décrit les bits système.

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Présentation des bits système

Généralités

Les automates Modicon M340, Premium, Atrium et Quantum disposent de bits système %Si qui indiquent les états de l'automate ou permettent d'agir sur le fonctionnement de celui-ci

Ces bits peuvent être testés dans le programme utilisateur afin de détecter toute évolution de fonctionnement devant entraîner une procédure particulière de traitement.

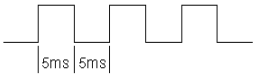
Certains de ces bits doivent être remis dans leur état initial ou normal par le programme. Cependant, les bits système qui sont remis dans leur état initial ou normal par le système ne doivent pas l'être par le programme ou par le terminal

Description des bits système %S0 à %S7

Description détaillée

Description des bits système %S0 à %S7 :

Bit Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum
%S0 COLDSTART	Démarrage à froid	Normalement sur 0, ce bit est réglé sur 1 par : ● une reprise secteur avec perte du contexte (défaut de la pile), ● le programme utilisateur, ● le bornier, ● un changement de cartouche. Ce bit est réglé sur 1 lors du premier cycle de restauration complet de l'automate en mode RUN ou STOP. Il est remis à 0 par le système avant le cycle suivant. Pour détecter le premier cycle en RUN après un démarrage à froid, reportez-vous à %SW10. En mode sûr, ce bit n'est pas disponible sur les automates de sécurité Quantum. %S0 n'est pas toujours réglé sur 1 lors du premier balayage de l'automate. Si le signal doit être réglé sur 1 à chaque démarrage de l'automate, il vaut mieux utiliser %S21 à la place. Pour Premium et Quantum, voir Traitement lors d'un démarrage à froid des automates Premium/Quantum *(voir page 139)* Pour Modicon M340, voir Traitement lors d'un démarrage à froid pour les automates Modicon M340 *(voir page 129)*	1 (1 cycle)	OUI	OUI	OUI
%S1 WARMSTART	Démarrage à chaud	Normalement sur 0, ce bit est réglé sur 1 par : ● une reprise secteur avec enregistrement de données, ● le programme utilisateur, ● le bornier. Il est remis à 0 par le système à la fin du premier cycle complet et avant la mise à jour des sorties.	0	OUI	OUI	OUI (sauf pour les automates de sécurité)

Bit Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum
		Ce bit n'est pas disponible sur les automates de sécurité Quantum. %S1 n'est pas toujours réglé sur 1 lors du premier balayage de l'automate. Si le signal doit être réglé sur 1 à chaque démarrage de l'automate, il vaut mieux utiliser %S21 à la place.				
%S4 TB10MS	Base de temps 10 ms	Un temporisateur interne régule le changement d'état de ce bit. Il est asynchrone par rapport au cycle de l'automate. Graphique :  Ce bit n'est pas disponible sur les automates de sécurité Quantum.	-	OUI	OUI	OUI (sauf pour les automates de sécurité)
%S5 TB100MS	Base de temps 100 ms	Idem %S4	-	OUI	OUI	OUI (sauf pour les automates de sécurité)
%S6 TB1SEC	Base de temps 1 s	Idem %S4	-	OUI	OUI	OUI (sauf pour les automates de sécurité)
%S7 TB1MIN	Base de temps 1 min	Idem %S4	-	OUI	OUI	OUI (sauf pour les automates de sécurité)

Description des bits système %S9 à %S13

Description détaillée

Description des bits système %S9 à %S13 :

Bit Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum
%S9 OUTDIS	Mise en position de repli des sorties sur tous les bus	Normalement à l'état 0, est mis à l'état 1 par le programme ou par le terminal : ● valeur 1 : provoque la mise à 0 ou le maintien de la valeur suivant le choix effectué en configuration (bus X, Fipio, AS-i, etc.). ● valeur 0 : les sorties sont mises à jour normalement. **Remarque :** le bit système agit directement sur les sorties physiques et non sur les bits images des sorties. **Remarque :** sur Modicon M340, le scrutateur I/O Ethernet et la fonction Global Data sont affectés par le bit %S9. **(1) Remarque :** sur Modicon M340, les entrées/sorties distribuées via le bus CANopen ne sont pas affectées par le bit %S9. Sur Modicon M340, après un mode d'exploitation, les sorties sont en mode sécurité (état égal à 0) pendant que le bit est défini.	0	OUI (1)	OUI	NON
%S10 IOERR	Défaut d'entrées/ sorties	Normalement à l'état 1, ce bit est réglé sur 0 quand un défaut sur un module en rack ou un équipement sur Fipio est détecté (configuration non conforme, défaut d'échange, défaut matériel, etc.). Le bit %S10 est remis à 1 par le système dès la disparition du défaut.	1	OUI	OUI	OUI

⚠ ATTENTION

COMPORTEMENT INATTENDU DE L'APPLICATION - COMPORTEMENT SPECIFIQUE DE VARIABLES

Sur Quantum, les erreurs de communication réseau avec des équipements distants qui sont détectées par les modules de communication (NOM, NOE, NWM, CRA, CRP) et les modules de commande de mouvement (MMS) ne sont pas signalées sur les bits %S10, %S16 et %S119.

Le non-respect de ces instructions peut provoquer des blessures ou des dommages matériels.

Bit Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum
%S11 WDG	Dépassements du chien de garde	Normalement à l'état 0, est mis à l'état 1 par le système dès que le temps d'exécution d'une tâche devient supérieur au temps d'exécution maximum (chien de garde) déclaré dans les propriétés de la tâche.	0	OUI	OUI	OUI
%S12 PLCRUNNING	Automate en RUN	Ce bit est mis à l'état 1 par le système lorsque l'automate est en RUN. Il est réglé sur 0 par le système dès que l'automate n'est plus en mode RUN (état STOP, INIT, etc.).	0	OUI	OUI	OUI
%S13 1RSTSCANRUN	Premier cycle après mise en RUN	Le passage de l'automate du mode STOP au mode RUN (y compris après un démarrage à froid) est signalé par la mise à 1 du bit système %S13. Ce bit est remis à 0 à la fin du premier cycle de la tâche MAST en mode RUN.	-	OUI	OUI	OUI

Description des bits système %S15 à %S21

Description détaillée

Description des bits système %S15 à %S21 :

Bit Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum
%S15 STRINGERROR	Défaut chaîne de caractères	Normalement à l'état 0, ce bit est mis à l'état 1 quand la zone de destination d'un transfert de chaîne de caractères n'a pas la taille suffisante (comprenant le nombre de caractères et le caractère de fin de chaîne de caractères) pour accueillir cette chaîne de caractères. L'application s'arrête en erreur si le bit %S78 a été mis à 1. Ce bit doit être remis à 0 par l'application. Ce bit n'est pas disponible sur les automates de sécurité Quantum.	0	OUI	OUI	OUI (sauf pour les automates de sécurité)
%S16 IOERRTSK	Défaut d'entrées/ sorties tâche	Normalement à l'état 1, ce bit est réglé sur 0 par le système quand un défaut sur un module en rack ou un équipement sur Fipio est détecté (configuration non conforme, défaut d'échange, défaut matériel, etc.). Ce bit doit être remis à 1 par l'utilisateur.	1	OUI	OUI	OUI

⚠ ATTENTION

COMPORTEMENT INATTENDU DE L'APPLICATION - COMPORTEMENT SPECIFIQUE DE VARIABLES

Sur Quantum, les erreurs de communication réseau avec des équipements distants qui sont détectées par les modules de communication (NOM, NOE, NWM, CRA, CRP) et les modules de commande de mouvement (MMS) ne sont pas signalées sur les bits %S10, %S16 et %S119.

Le non-respect de ces instructions peut provoquer des blessures ou des dommages matériels.

Bit Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum
%S17 CARRY	Sortie décalage circulaire	Normalement à l'état 0. Lors d'une opération de décalage circulaire, ce bit prend l'état du bit sortant.	0	OUI	OUI	OUI
%S18 OVERFLOW	Dépassement ou erreur arithmétique	Normalement à l'état 0, ce bit est mis à l'état 1 en cas de dépassement de capacité dans les cas suivants : ● un résultat supérieur à + 32 767 ou inférieur à - 32 768, en mono-longueur, ● un résultat supérieur à + 65 535, en entier non signé, ● un résultat supérieur à + 2 147 483 647 ou inférieur à - 2 147 483 648, en double-longueur, ● un résultat supérieur à +4 294 967 296, en double-longueur ou en entier non signé, ● valeurs réelles hors bornes, ● division par 0, ● racine d'un nombre négatif, ● forçage à un pas inexistant sur un programmateur cyclique, ● empilage d'un registre plein, dépilage d'un registre vide. Il n'existe qu'un seul cas où le bit %S18 n'est pas augmenté par les automates Modicon M340 lorsque des valeurs réelles sont hors bornes. C'est lorsque des opérandes non normalisés ou certaines opérations générant des résultats non normalisés sont utilisés (dépassement progressif par valeur inférieure). Doit être testé par le programme utilisateur après chaque opération présentant un risque de dépassement, puis remis à 0 par l'utilisateur en cas de dépassement. Lorsque le bit %S18 passe à 1, l'application s'arrête en erreur si le bit %S78 a été réglé sur 1.	0	OUI	OUI	OUI

Bit Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum
%S19 OVERRUN	Dépassement période de tâche (scrutation périodique)	Normalement à l'état 0, ce bit est réglé sur 1 par le système en cas de dépassement de la période d'exécution (temps d'exécution de tâche supérieur à la période définie par l'utilisateur dans la configuration ou programmée dans le mot %SW associé à la tâche). Ce bit doit être remis à 0 par l'utilisateur. Chaque tâche gère son propre bit %S19.	0	OUI	OUI	OUI
%S20 INDEXOVF	Dépassement d'index	Normalement à l'état 0, ce bit est réglé sur 1 lorsque l'adresse de l'objet indexé devient inférieure à 0 ou dépasse le nombre d'objets déclaré dans la configuration. Dans ce cas, l'index est considéré comme étant égal à 0. Doit être testé par le programme utilisateur, après chaque opération où il y a risque de dépassement, puis remis à 0 en cas de dépassement. Lorsque le bit %S20 passe à 1, l'application s'arrête en erreur si le bit %S78 a été mis à 1. Ce bit n'est pas disponible sur les automates de sécurité Quantum.	0	OUI	OUI	OUI (sauf pour les automates de sécurité)
%S21 1RSTTASKRUN	Premier cycle de tâche	Testé dans une tâche (Mast, Fast, Aux0, Aux1, Aux2, Aux3), le bit %S21 indique le premier cycle de cette tâche, y compris après un démarrage à froid avec démarrage automatique en mode Run et un démarrage à chaud. %S21 est mis à 1 en début de cycle et remis à zéro en fin de cycle. **Remarque :** le bit %S21 ne possède pas la même signification sous PL7 et sous Unity Pro.	0	OUI	OUI	OUI

Description des bits système %S30 à %S59

Description détaillée

Description des bits système %S30 à %S59 :

Bit Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum
%S30 MASTACT	Activation/ désactivation de la tâche maître	Normalement à l'état 1. La tâche maître est désactivée lorsque l'utilisateur règle le bit sur 0. Ce bit est pris en compte par le système à la fin de chaque cycle de tâche MAST. Ce bit n'est pas disponible sur les automates de sécurité Quantum.	1	OUI	OUI	OUI (sauf pour les automates de sécurité)
%S31 FASTACT	Activation/ désactivation de la tâche rapide	Normalement réglé sur 1 lorsque l'utilisateur crée la tâche. La tâche est désactivée lorsque l'utilisateur règle le bit sur 0. Ce bit n'est pas disponible sur les automates de sécurité Quantum.	1	OUI	OUI	OUI (sauf pour les automates de sécurité)
%S32 AUX0ACT	Activation/ désactivation de la tâche auxiliaire 0	Normalement réglé sur 1 lorsque l'utilisateur crée la tâche. La tâche auxiliaire est désactivée lorsque l'utilisateur règle le bit sur 0. Ce bit n'est pas disponible sur les automates de sécurité Quantum.	0	NON	OUI	OUI (sauf pour les automates de sécurité)
%S33 AUX1ACT	Activation/ désactivation de la tâche auxiliaire 1	Normalement réglé sur 1 lorsque l'utilisateur crée la tâche. La tâche auxiliaire est désactivée lorsque l'utilisateur règle le bit sur 0. Ce bit n'est pas disponible sur les automates de sécurité Quantum.	0	NON	OUI	OUI (sauf pour les automates de sécurité)
%S34 AUX2ACT	Activation/ désactivation de la tâche auxiliaire 2	Normalement réglé sur 1 lorsque l'utilisateur crée la tâche. La tâche auxiliaire est désactivée lorsque l'utilisateur règle le bit sur 0. Ce bit n'est pas disponible sur les automates de sécurité Quantum.	0	NON	OUI	OUI (sauf pour les automates de sécurité)
%S35 AUX3ACT	Activation/ désactivation de la tâche auxiliaire 3	Normalement réglé sur 1 lorsque l'utilisateur crée la tâche. La tâche auxiliaire est désactivée lorsque l'utilisateur règle le bit sur 0. Ce bit n'est pas disponible sur les automates de sécurité Quantum.	0	NON	OUI	OUI (sauf pour les automates de sécurité)

Bit **Symbole**	**Fonction**	**Description**	**Etat initial**	**Modicon M340**	**Premium Atrium**	**Quantum**
%S38 ACTIVEVT	Activation/ inhibition des événements	Normalement réglé sur 1. Les événements sont inhibés lorsque l'utilisateur règle le bit sur 0. Ce bit n'est pas disponible sur les automates de sécurité Quantum.	1	OUI	OUI	OUI (sauf pour les automates de sécurité)
%S39 EVTOVR	Saturation lors du traitement de l'événement	Ce bit est réglé sur 1 par le système pour indiquer qu'un ou plusieurs événement ne peuvent pas être traités lorsque les files d'attente sont saturées. Ce bit est remis à l'état 0 par l'utilisateur. Ce bit n'est pas disponible sur les automates de sécurité Quantum.	0	OUI	OUI	OUI (sauf pour les automates de sécurité)
%S40 RACK0ERR	Défaut d'entrée/de sortie du rack 0	Le bit %S40 est attribué au rack 0. Normalement réglé sur 1. Ce bit prend la valeur 0 en cas de défaillance des entrées/sorties sur le rack. Dans ce cas : ● le bit %S10 est réglé sur 0, ● le voyant du processeur des E/S est allumé, ● le module %Ir.m.c.Err est réglé sur 1. Ce bit reprend la valeur 1 lorsque la défaillance disparaît.	1	OUI	OUI	NON
%S41 RACK1ERR	Défaut d'entrée/de sortie du rack 1	Identique à %S40 pour le rack 1.	1	OUI	OUI	NON
%S4 2 RACK2ERR	Défaut d'entrée/de sortie du rack 2	Identique à %S40 pour le rack 2.	1	OUI	OUI	NON
%S43 RACK3ERR	Défaut d'entrée/de sortie du rack 3	Identique à %S40 pour le rack 3.	1	OUI	OUI	NON
%S44 RACK4ERR	Défaut d'entrée/de sortie du rack 4	Identique à %S40 pour le rack 4.	1	OUI	OUI	NON
%S45 RACK5ERR	Défaut d'entrée/de sortie du rack 5	Identique à %S40 pour le rack 5.	1	OUI	OUI	NON

Bit Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum
%S46 RACK6ERR	Défaut d'entrée/de sortie du rack 6	Identique à %S40 pour le rack 6.	1	OUI	OUI	NON
%S47 RACK7ERR	Défaut d'entrée/de sortie du rack 7	Identique à %S40 pour le rack 7.	1	OUI	OUI	NON
%S50 RTCWRITE	Mise à jour de la date et de l'heure à l'aide des mots %SW50 à %SW53	Normalement réglé sur 0. Ce bit est réglé sur 1 ou 0 par le programme ou le bornier. ● valeur 0 : mise à jour des mots système %SW50 à %SW53 avec la date et l'heure fournies par l'horodateur de l'automate. ● valeur 1 : les mots système %SW50 à %SW53 ne sont plus mis à jour, ce qui rend possible leur modification. ● Le basculement de 1 à 0 met à jour l'horodateur avec les valeurs saisies dans les mots %SW50 à %SW53.	0	OUI	OUI	OUI
%S51 RTCERR	Retard de l'horodateur	Lorsqu'il est réglé sur 1, ce bit géré par le système indique que l'horodateur est manquant ou que ses mots système (%SW50 à %SW53) sont sans signification. Dans ce cas, l'horodateur doit être réinitialisé sur l'heure correcte.	–	OUI	OUI	OUI
%S59 RTCTUNING	Mise à jour incrémentale de la date et de l'heure à l'aide du mot %SW59	Normalement réglé sur 0. Ce bit est réglé sur 1 ou 0 par le programme ou le bornier : ● valeur 0 : le système ne gère pas le mot système %SW59, ● valeur 1 : le système gère les fronts sur le mot %SW59 pour régler la date et l'heure courantes (par incrément).	0	OUI	OUI	OUI

Description des bits système %S60 à %S79

Description détaillée

Description des bits système %S60 à %S79 :

Bit Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum
%S60 BACKUPCHVOV	Commande de commutation volontaire	Ce bit commande la commutation volontaire en cas de mise en œuvre d'une architecture redondante. Il peut être réinitialisé à 0 par l'utilisateur ou par l'application. Disponible uniquement pour Premium. Par défaut, ce bit est réglé sur 0 ; s'il est réglé sur 1, il ne se produit aucun basculement.	0	NON	OUI	NON
%S65 CARDIS	Désactivation de carte	Il est nécessaire de générer un front montant sur le bit %S65 avant d'extraire la carte, afin de garantir la cohérence des informations. En fait, au moment de la détection du front montant, les accès en cours prennent fin (lecture et modification de fichiers, enregistrement d'application) et le voyant d'accès à la carte est éteint (l'état du voyant CARDERR reste inchangé). Vous pouvez ensuite retirer la carte ; le voyant CARDERR est alors allumé. Insertion de la carte : le voyant d'accès est allumé et le voyant CARDERR précise l'état (%S65 reste inchangé). L'utilisateur doit réinitialiser %S65 à 0 pour permettre la détection ultérieure du front montant. Si un front montant a été généré sur le bit %S65 mais que la carte n'ait pas été extraite, la remise à 0 du bit n'est pas suffisante pour pouvoir accéder à la carte.	0	OUI	NON	NON

Bit Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum
%S66 LEDBATT APPLIBCK	Sauvegarde de l'application	L'utilisateur paramètre ce bit sur 1 pour démarrer une opération de sauvegarde (transfert de l'application de la RAM à la carte). Le système détectera le front montant pour démarrer la sauvegarde. Il procède chaque seconde à des interrogations pour connaître l'état de ce bit. Une sauvegarde a lieu uniquement si l'application dans la RAM est différente de celle sur la carte. Une fois la sauvegarde terminée, le système paramètre ce bit sur 0. **Avertissement :** avant d'effectuer une nouvelle sauvegarde en réglant le bit %S66 sur 1, vous devez vérifier que celui-ci avait été réglé sur 0 par le système (ce qui signifie que la sauvegarde précédente est terminée). N'utilisez jamais %S66 s'il a la valeur 1. Vous risqueriez de perdre des données. Le bit %S66 est particulièrement utile après le remplacement de la valeur d'initialisation %S94 et l'enregistrement de paramètres.	0	OUI	NON	NON

Bit Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum
%S67 PCMCIABAT0	Etat pile carte mémoire application	Ce bit est utilisé pour surveiller l'état de la pile principale quand la carte mémoire est dans l'emplacement PCMCIA **supérieur**. Cela s'applique aux automates Atrium, Premium et Quantum (CPU 140 CPU 671 60/60S, 140 CPU 651 60/60S, 140 CPU 652 60 et 140 CPU 651 50) : ● valeur 1 : tension faible sur la pile principale. L'application est conservée mais la pile doit être remplacée à la suite de la procédure de maintenance prédictive *(voir Premium et Atrium sous Unity Pro, Processeurs, racks et alimentations, Manuel de mise en oeuvre)*), ● valeur 0 : tension suffisante sur la pile principale (application toujours conservée). ● Le bit %S67 est pris en charge par Unity version ≥ 2.02. **NOTE :** avec les cartes PCMCIA bleues (PV >= 04), le bit %S67 n'est pas réglé sur 1 lorsque la pile principale est absente, alors qu'il l'est dans les mêmes circonstances avec les cartes PCMCIA vertes (PV <04).	-	NON	OUI	OUI
%S68 PLCBAT	Etat pile processeur	Ce bit permet de contrôler l'état de fonctionnement de la pile de sauvegarde pour l'enregistrement des données et du programme en mémoire RAM. ● valeur 0 : pile présente et en service ● valeur 1 : pile absente ou hors service	-	NON	OUI	OUI

Bit Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum
%S75 PCMCIABAT1	Etat pile carte mémoire stockage de données	Ce bit est pris en charge à partir de la version 2.02 de Unity Pro. Il est utilisé pour surveiller l'état de la pile principale quand la carte mémoire est dans l'emplacement PCMCIA **inférieur**. %S75 est pris en charge par les processeurs Premium suivants : TSX P57 4••, TSX P57 5•• et TSX P57 6••. **NOTE :** pour tous les autres processeurs Premium, %S75 n'indique un niveau de pile faible qu'une fois que la pile a déjà atteint un niveau critique. %S75 est pris en charge par les processeurs Quantum suivants : 140 CPU 671 60/60S*, 140 CPU 651 60/60S*, 140 CPU 652 60 et 140 CPU 651 50. %S75 est : ● réglé sur 1 quand la tension de la pile principale est faible. L'application est conservée mais la pile doit être remplacée à la suite de la procédure de maintenance prédictive *(voir Premium et Atrium sous Unity Pro, Processeurs, racks et alimentations, Manuel de mise en oeuvre)* ; ● réglé sur 0 quand la tension de la pile principale est suffisante (l'application est toujours conservée). ***** Les données stockées sur une carte mémoire située à l'emplacement B ne sont pas traitées dans le cadre des projets de sécurité.	-	NON	OUI	OUI
%S76 DIAGBUFFCONF	Tampon de diagnostic configuré	Ce bit est réglé sur 1 par le système lorsque l'option de diagnostic est configurée : un tampon de diagnostic destiné au stockage des erreurs issues des DFB de diagnostic est alors réservé. Ce bit est en lecture seule.	0	OUI	OUI	OUI

Bit **Symbole**	**Fonction**	**Description**	**Etat initial**	**Modicon M340**	**Premium Atrium**	**Quantum**
%S77 DIAGBUFFFFULL	Tampon de diagnostic plein	Ce bit est réglé sur 1 par le système lorsque le tampon recevant les erreurs des blocs fonction de diagnostic est rempli. Ce bit est en lecture seule.	0	OUI	OUI	OUI
%S78 HALTIFERROR	Arrêt si erreur	Normalement à 0, ce bit peut être réglé sur 1 par l'utilisateur, pour programmer l'arrêt de l'automate en cas de dysfonctionnement de l'application : %S15, %S18, %20. Sur les automates de sécurité Quantum, l'état Pause est remplacé par l'état Erreur en mode sûr. Par ailleurs, %S15 et %20 ne sont pas disponibles.	0	OUI	OUI	OUI
%S79 MBFBCTRL	Contrôle des bits forcés Modbus	Ce bit modifie le comportement du serveur Quantum Modbus face aux bits forcés : ● s'il a la valeur 0 (par défaut), la gestion est standard : cette valeur est modifiée même si le bit est forcé. ● si l'utilisateur le paramètre sur 1 : les demandes d'écriture de bits forcés n'entraînent pas la modification de leur valeur. Toutefois, aucune erreur n'est indiquée dans la réponse à la demande. Le bit d'historique est continuellement mis à jour, quel que soit l'état de forçage.	0	NON	NON	OUI

Description des bits système %S80 à %S96

Description détaillée

Description des bits système %S80 à %S96 :

Bit Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum
%S80 RSTMSGCNT	Remise à zéro des compteurs de messages	Normalement à l'état 0, ce bit peut être mis à 1 par l'utilisateur pour remettre à zéro les compteurs de messages %SW80 à %SW86.	0	OUI	OUI	OUI
%S82	Réglage de l'interrogation MB+PCMCIA	Ce bit est utilisé pour modifier le mode d'échange de requête avec l'équipement MB+MBP100 PCMCIA. Par défaut (valeur 0), le système envoie une requête à la carte et l'interroge pour obtenir la réponse au cours du cycle Mast suivant. Ce mode est conseillé pour les cycles Mast courts. Lorsqu'il est réglé sur 1, le système envoie une requête à la carte et attend une réponse. Ce mode est conseillé en cas de cycle Mast long.	0	NON	OUI	NON
%S90 COMRFSH	Actualisation des mots communs	Normalement à l'état 0, ce bit est réglé sur 1 à la réception de mots communs venant d'une autre station du réseau. Ce bit peut être réglé sur 0 par le programme ou par le terminal pour vérifier le cycle d'échange des mots communs.	0	NON	OUI	NON
%S91 LCKASYNREQ	Verrouillage requête asynchrone	Si ce bit est réglé sur 1, les requêtes de communication asynchrones traitées dans la tâche surveillance sont exécutées intégralement sans interruption des autres tâches comme la MAST ou FAST. Ainsi, la lecture ou l'écriture cohérente des données est garantie. **Rappel** : le serveur de requête de la tâche surveillance est appelé via la porte 7 (X-Way).	0	NON	OUI	NON

Bit **Symbole**	**Fonction**	**Description**	**Etat initial**	**Modicon M340**	**Premium Atrium**	**Quantum**
%S92 EXCHGTIME	Mode de mesure de la fonction de communication	Normalement à l'état 0, ce bit peut être réglé sur 1 par l'utilisateur pour positionner les fonctions de communication en mode de mesure des performances. Le paramètre de timeout des fonctions de communication (qui figure dans le tableau de gestion) affiche alors le temps d'échange aller/retour en millisecondes. **Remarque** : les fonctions de communication sont exécutées sur la base de temps de 100 ms.	0	OUI	OUI	NON
%S94 SAVECURRVAL	Enregistrement des valeurs de réglage	Normalement à l'état 0, ce bit peut être réglé sur 1 par l'utilisateur pour remplacer les valeurs initiales des variables déclarées avec un attribut « Save » (par exemple : variables DFB) par les valeurs courantes. Pour Modicon M340, sur un front montant %S94, le contenu de la RAM interne et celui de la carte mémoire sont différents (%S96 = 0 et le voyant CARDERR est allumé). Lors d'un démarrage à froid, les valeurs en cours sont remplacées par les valeurs initiales les plus récentes, à condition qu'un enregistrement sur carte mémoire ait été exécuté auparavant (fonction Enregistrer la sauvegarde ou front montant %S66). Le système remet le bit %S94 à 0 quand le remplacement est terminé. Remarque : ce bit doit être utilisé avec précaution : ne le réglez pas de façon définitive sur 1 et employez uniquement la tâche maître. Ce bit n'est pas disponible sur les automates de sécurité Quantum. En cas d'utilisation avec la mémoire flash PCMCIA TSX MFP • ou TSX MCP •, l'enregistrement des valeurs de réglage n'est pas disponible.	0	OUI	OUI	OUI (sauf pour les automates de sécurité)

Bit Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum
%S96 BACKUPPROGOK	Programme de sauvegarde OK	Ce bit est réglé sur 0 ou 1 par le système. ● Il est réglé sur 0 lorsque la carte est manquante ou inutilisable (mauvais format, type non reconnu, etc.) ou lorsque son contenu ne correspond pas à la RAM de l'application interne. ● Il est réglé sur 1 lorsque la carte fonctionne et que son contenu correspond à la RAM de l'application interne de l'UC.	-	OUI	NON	NON

⚠ ATTENTION

ECHEC DU CHARGEMENT DE L'APPLICATION

Le bit %S94 ne doit pas être réglé sur 1 lors d'un chargement.

Si le bit %S94 est réglé sur 1, le chargement risque d'être impossible.

Le non-respect de ces instructions peut provoquer des blessures ou des dommages matériels.

⚠ ATTENTION

PERTE DE DONNEES

Le bit %S94 doit être utilisé avec la mémoire flash PCMCIA TSX MFP • ou TSX MCP •. La fonction de ce bit système n'est pas disponible avec ce type de mémoire.

Le non-respect de ces instructions peut provoquer des blessures ou des dommages matériels.

Description des bits système %S100 à % S123

Description détaillée

Description des bits système %S100 à % S123 :

Bit Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum
%S100 PROTTERINL	Protocole sur prise terminal	Ce bit est paramétré sur 0 ou 1 par le système suivant l'état de la dérivation INL/DPT sur la prise console. ● Si la dérivation est absente (%S100 = 0), le protocole Uni-Telway maître est utilisé. ● Si la dérivation est présente (%S100 = 1), le protocole utilisé est celui indiqué dans la configuration de l'application.	-	NON	OUI	NON
%S118 REMIOERR	Défaut général E/S distantes	Normalement à 1, ce bit est réglé sur 0 par le système lors de l'apparition d'un défaut sur un équipement connecté aux bus d'entrées/de sorties distantes RIO (Fipio pour Premium ou station S908 pour Quantum). Ce bit reprend la valeur 1 lorsque l'erreur disparaît. Ce bit n'est pas mis à jour si une erreur se produit sur les autres bus (DIO, ProfiBus, ASI).	-	OUI	OUI	OUI
%S119 LOCIOERR	Défaut général E/S en rack	Normalement à 1, ce bit est mis à 0 par le système lors de l'apparition d'une erreur sur un module d'E/S installé dans l'un des racks. Ce bit reprend la valeur 1 lorsque l'erreur disparaît.	-	OUI	OUI	OUI

⚠ ATTENTION

%S119 pour automates Quantum

Sur Quantum, les erreurs de communication réseau avec des équipements distants qui sont détectées par les modules de communication (NOM, NOE, NWM, CRA, CRP) et les modules de commande de mouvement (MMS) ne sont pas signalées sur les bits %S10, %S16 et %S119.

Le non-respect de ces instructions peut provoquer des blessures ou des dommages matériels.

Bit **Symbole**	**Fonction**	**Description**	**Etat initial**	**Modicon M340**	**Premium Atrium**	**Quantum**
%S120 DIOERRPLC	Défaut bus DIO (UC)	Normalement à 1, ce bit est mis à 0 par le système lors de l'apparition d'une erreur sur un équipement connecté au bus DIO géré par la liaison Modbus Plus intégrée à l'UC. Ce bit n'est pas disponible sur les automates de sécurité Quantum. Certaines informations sont disponibles dans le Viewer de diagnostic (si l'entrée est sélectionnée) afin de clarifier le type d'erreur sur le bus. Ces informations peuvent déterminer le bus déporté correct avec le numéro de bus (RIO, DIO).	-	NON	NON	OUI (sauf pour les automates de sécurité)
%S121 DIOERRNOM1	Défaut bus DIO (NOM n° 1)	Normalement à 1, ce bit est mis à 0 par le système lors de l'apparition d'une erreur sur un équipement connecté au bus DIO géré par le premier module 140 NOM 2••. Ce bit n'est pas disponible sur les automates de sécurité Quantum. Certaines informations sont disponibles dans le Viewer de diagnostic (si l'entrée est sélectionnée) afin de clarifier le type d'erreur sur le bus. Ces informations peuvent déterminer le bus déporté correct avec le numéro de bus (RIO, DIO).	-	NON	NON	OUI (sauf pour les automates de sécurité)
%S122 DIOERRNOM2	Défaut bus DIO (NOM n° 2)	Normalement à 1, ce bit est mis à 0 par le système lors de l'apparition d'une erreur sur un équipement connecté au bus DIO géré par le second module 140 NOM 2••. Ce bit n'est pas disponible sur les automates de sécurité Quantum. Certaines informations sont disponibles dans le Viewer de diagnostic (si l'entrée est sélectionnée) afin de clarifier le type d'erreur sur le bus. Ces informations peuvent déterminer le bus déporté correct avec le numéro de bus (RIO, DIO).	-	NON	NON	OUI (sauf pour les automates de sécurité)
%S123 ADJBX	Réglage bus X	Ce bit est utilisé par le système et ne peut pas être utilisé par l'application utilisateur.	-	OUI	OUI	NON

6.2 Mots système

Objet de cette section

Ce chapitre décrit les mots système Modicon M340, Atrium, Premium et Quantum.

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Description des mots système %SW0 à %SW11

Description détaillée

Description des mots système %SW0 à %SW11.

Mot Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum
%SW0 MASTPERIOD	Période de scrutation de la tâche MAST	Ce mot est utilisé pour modifier la période de la tâche Mast via le programme utilisateur ou via le bornier. La période est exprimée en ms (1...255 ms) %SW0=0 en fonctionnement cyclique. Lors d'un redémarrage à froid : ce mot prend la valeur définie dans la configuration. Ce mot n'est pas disponible sur les automates de sécurité Quantum.	0	OUI	OUI	OUI (sauf pour les automates de sécurité)
%SW1 FASTPERIOD	Période de scrutation de la tâche FAST	Ce mot est utilisé pour modifier la période de la tâche Fast via le programme utilisateur ou via le bornier. Cette période est exprimée en millisecondes (1 à 255 ms). Lors d'un redémarrage à froid, ce mot prend la valeur définie dans la configuration. Ce mot n'est pas disponible sur les automates de sécurité Quantum.	0	OUI	OUI	OUI (sauf pour les automates de sécurité)
%SW2 AUX0PERIOD **%SW3** AUX1PERIOD **%SW4** AUX2PERIOD **%SW5** AUX3PERIOD	Période de scrutation de la tâche auxiliaire	Ce mot est utilisé pour modifier la période des tâches définies dans la configuration, via le programme utilisateur ou via le bornier. La période est exprimée en dizaine de ms (10 ms à 2,55 s). (1) uniquement sur les automates 140 CPU 6•• et TSX 57 5••. Ces mots ne sont pas disponibles sur les automates de sécurité Quantum.	0	NON	OUI (1)	OUI (1) (sauf pour les automates de sécurité)
%SW6 **%SW7**	Adresse IP	Adresse IP du port Ethernet de l'UC. La modification de la valeur de ce mot n'est pas prise en compte. Ce bit est sur 0 lorsque l'UC ne dispose pas de port Ethernet.	-	OUI	NON	NON

Mot Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum
%SW8 TSKINHIBIN	Surveillance de l'acquisition d'entrées de tâche	Normalement à l'état 0, ce bit peut être paramétré sur 1 ou 0 par le programme ou le terminal. Il inhibe la phase d'acquisition d'entrées de chaque tâche : ● %SW8.0 = 1 inhibe l'acquisition d'entrées concernant la tâche MAST. ● %SW8.1 = 1 inhibe l'acquisition d'entrées concernant la tâche FAST. ● %SW8.2 à 5 = 1 inhibe l'acquisition d'entrées concernant les tâches auxiliaires AUX 0... 3. (1) **Remarque** : Sur Modicon M340, les entrées/sorties distribuées via le bus CANopen ne sont pas affectées par le mot %SW8. (2) **Remarque** : Sur Quantum, les entrées/sorties distribuées via le bus DIO ne sont pas affectées par le mot %SW8. Ce mot n'est pas disponible sur les automates de sécurité Quantum.	0	OUI (1)	OUI	OUI (2) (sauf pour les automates de sécurité)

Mot Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum
%SW9 TSKINHIBOUT	Surveillance de la mise à jour de sorties de tâches	Normalement à l'état 0, ce bit peut être paramétré sur 1 ou 0 par le programme ou le terminal. Il inhibe la phase de mise à jour de sorties de chaque tâche. ● %SW9.0 = 1 affecté à la tâche MAST ; les sorties relatives à cette tâche ne sont plus prises en charge. ● %SW9.1 = 1 affecté à la tâche FAST ; les sorties relatives à cette tâche ne sont plus prises en charge. ● %SW9.2 à 5 = 1 affecté aux tâches auxiliaires AUX 0... 3 ; les sorties relatives à ces tâches ne sont plus prises en charge. (3) **Remarque** : Sur Modicon M340, les entrées/sorties distribuées via le bus CANopen ne sont pas affectées par le mot %SW9. Sur Modicon M340, après un mode d'exploitation, les sorties sont en mode sécurité (état égal à 0) pendant que le bit est défini. (4) **Remarque** : Sur Quantum, les entrées/sorties distribuées via le bus DIO ne sont pas affectées par le mot %SW9. Ce mot n'est pas disponible sur les automates de sécurité Quantum.	0	OUI (3)	OUI	OUI (4) (sauf pour les automates de sécurité)

⚠ ATTENTION

RISQUES LIES A UN COMPORTEMENT INATTENDU

Sur Premium/Atrium :

Les sorties de modules situées sur le bus<:hs>X passent automatiquement en mode configuré (repli ou maintien). Sur le bus Fipio, certains équipements ne prennent pas en charge le mode repli. Dans ce cas, seul le mode maintien est possible.

Sur Quantum :

Toutes les sorties, ainsi que le rack local ou distant (RIO) sont maintenus à l'état précédant la mise à 1 du bit %SW9 correspondant à la tâche.

Les entrées/sorties distribuées (DIO) ne sont pas affectées par le mot système %SW9.

Le non-respect de ces instructions peut provoquer des blessures ou des dommages matériels.

Mot `Symbole`	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum
%SW10 `TSKINIT`	Premier cycle après un démarrage à froid	Si la valeur du bit de la tâche courante est définie sur 0, cela signifie que la tâche effectue son premier cycle après un démarrage à froid. ● `%SW10.0` : affecté à la tâche MAST. ● `%SW10.1` : affecté à la tâche FAST. ● `%SW10.2` à `5` : affecté aux tâches auxiliaires AUX 0... 3. Ce mot n'est pas disponible sur les automates de sécurité Quantum.	0	OUI	OUI	OUI (sauf pour les automates de sécurité)
%SW11 `WDGVALUE`	Durée du chien de garde	Lit la durée du chien de garde. La durée est exprimée en millisecondes (10...1 500 ms). Ce mot ne peut pas être modifié.	-	OUI	OUI	OUI

Description des mots système %SW12 à %SW29

Description détaillée

Description des mots système %SW12 à %SW29 :

Mot Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum
%SW12 UTWPORTADDR	Adresse du port série du processeur	Pour Premium : adresse Uni-Telway du port terminal (en mode esclave) définie dans la configuration et chargée dans ce mot lors d'un démarrage à froid. La modification de la valeur de ce mot n'est pas prise en compte par le système. Pour Modicon M340 : adresse esclave Modbus du port série de l'UC. La modification de la valeur de ce mot n'est pas prise en compte. Ce bit est mis à 0 lorsque l'UC ne dispose pas de port série.	-	OUI	OUI	NON (voir %SW12 ci-dessous)
%SW12 APMODE	Mode du processeur de l'application	Pour les automates de sécurité Quantum uniquement, ce mot indique le mode de fonctionnement du processeur d'application du module d'UC. • 16#A501 = mode de maintenance • 16#5AFE = mode sûr Toute autre valeur génère une erreur. **Remarque** : Avec un système de sécurité de redondance d'UC, ce mot est échangé entre l'automate primaire et l'automate redondant afin d'informer ce dernier de l'activation du mode sûr ou de maintenance.	16#A501	NON	NON	OUI Uniquement sur les automates de sécurité

Mot Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum
%SW13 XWAYNETWADDR	Adresse principale de la station	Ce mot fournit les données suivantes relatives au réseau principal (Fipway ou Ethway) : • le numéro de station (octet de poids faible), compris entre 0 et 127, • le numéro de réseau (octet de poids fort), compris entre 0 et 63, (la valeur des micro-interrupteurs sur la carte PCMCIA).	254 (16#00FE)	NON	OUI	NON (voir %SW13 ci-dessous)
%SW13 INTELMODE	Mode du processeur Intel	Pour les automates de sécurité Quantum uniquement, ce mot indique le mode de fonctionnement du processeur Intel Pentium du module d'UC. • 16#501A = mode de maintenance • 16#5AFE = mode sûr Toute autre valeur génère une erreur. **Remarque** : Avec un système de sécurité de redondance d'UC, ce mot est échangé entre l'automate primaire et l'automate redondant afin d'informer ce dernier de l'activation du mode sûr ou de maintenance.	-	NON	NON	OUI Uniquement sur les automates de sécurité
%SW14 OSCOMMVERS	Version commerciale du processeur de l'automate	Ce mot contient la version actuelle du système d'exploitation du processeur de l'automate. **Exemple :** 16#0135 version : 01 numéro de révision : 35	-	OUI	OUI	OUI
%SW15 OSCOMMPATCH	Version du patch processeur automate	Ce mot contient la version commerciale du patch pour le processeur automate. Le codage s'effectue sur l'octet de poids faible du mot. Codage : 0 = pas de patch, 1 = A, 2 = B... **Exemple :** 16#0003 correspond au patch C.	-	OUI	OUI	OUI

Mot Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum
%SW16 OSINTVERS	Numéro de version du micrologiciel	Ce mot contient le numéro de version en hexadécimal du micrologiciel du processeur de l'automate. **Exemple :** 16#0011 version : 2.1 numéro de révision : 17	-	OUI	OUI	OUI
%SW17 FLOATSTAT	Etat d'erreur sur opération flottante	Lorsqu'une erreur est détectée dans une opération arithmétique flottante, le bit %S18 est réglé sur 1 et l'état d'erreur du mot %SW17 est mis à jour selon le codage suivant : • %SW17.0 = opération non valide/le résultat n'est pas un nombre, • %SW17.1 = opérande non normalisé/résultat acceptable (indicateur non géré par Modicon M340 ou les automates de sécurité Quantum), • %SW17.2 = division par 0 / le résultat est l'infini, • %SW17.3 = dépassement / le résultat est l'infini, • %SW17.4 = dépassement par valeur inférieure/le résultat est 0, • %SW17.5 à 15 = non utilisés. Ce mot est remis à 0 par le système lors d'un démarrage à froid, mais aussi par le programme pour pouvoir être réutilisé. Ce mot n'est pas disponible sur les automates de sécurité Quantum.	0	OUI	OUI	OUI Uniquement sur les automates de sécurité

Mot Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum
%SD18 **%SW18** et **%SW19** 100MSCOUNTER	Compteur de temps absolu	%SW18 représente les octets de poids faible et %SW19 les octets de poids fort du double mot %SD18, qui est incrémenté par le système chaque 1/10e de seconde. L'application peut lire ou écrire ces mots afin d'effectuer des calculs de durée. %SD18 est incrémenté systématiquement, même en mode STOP et dans des états équivalents. Cependant, les durées au cours desquelles l'automate est éteint ne sont pas prises en compte (fonction non liée au programmateur en temps réel, mais uniquement à l'horodateur). Pour les automates de sécurité Quantum, sachant que les 2 processeurs doivent traiter exactement les mêmes données, la valeur de %SD18 est mise à jour au début de la tâche maître, puis reste fixe tout au long de l'exécution de l'application.	0	OUI	OUI	OUI

Mot **Symbole**	**Fonction**	**Description**	**Etat initial**	**Modicon M340**	**Premium Atrium**	**Quantum**
%SD20 : %SW20 et **%SW21** MSCOUNTER	Compteur de temps absolu	Pour les automates M340 et Quantum, %SD20 est incrémenté tous les 1/1 000e de seconde par le système (même lorsque l'automate est en mode STOP ; %SD20 n'est plus incrémenté si l'automate est éteint). %SD20 peut être lu par le programme utilisateur ou par le terminal. %SD20 est réinitialisé lors d'un démarrage à froid. %SD20 n'est pas réinitialisé lors d'une reprise à chaud. Dans le cas des automates Premium TSX P57 1•4M/2•4M/3•4M/C024 M/024M et TSX PCI57 204M/354M, %SD20 est incrémenté de 5 tous les 5/1 000e de seconde par le système. Pour tous les autres automates Premium, le compteur de temps de %SD20 est paramétré sur 1 ms comme pour les automates Quantum et M340. Pour les automates de sécurité Quantum, sachant que les 2 processeurs doivent traiter exactement les mêmes données, la valeur de %SD18 est mise à jour au début de la tâche maître, puis reste fixe tout au long de l'exécution de l'application.	0	OUI	OUI	OUI
%SW23	Valeur du commutateur rotatif	L'octet de poids faible contient le commutateur rotatif du processeur Ethernet. Il peut être lu par le programme utilisateur ou par le terminal.	-	OUI	NON	NON

Mot **Symbole**	**Fonction**	**Description**	**Etat initial**	**Modicon M340**	**Premium Atrium**	**Quantum**
%SW26	Nombre de requêtes traitées	Ce mot système permet de vérifier, côté serveur, le nombre de requêtes traitées par l'automate à chaque cycle.	-	OUI	OUI	OUI
%SW27 **%SW28** **%SW29**	Durée surdébit	• %SW27 contient la dernière durée du surdébit. • %SW28 contient la durée maximum du surdébit. • %SW29 contient la durée minimum du surdébit. La durée du surdébit dépend de la configuration (nombre d'E/S, etc.) et des requêtes de cycle en cours (communication, diagnostic). Durée du surdébit = temps de cycle Mast - temps d'exécution du code utilisateur. Ces mots peuvent être lus et écrits par le programme utilisateur ou par le terminal.	-	OUI	NON	NON

Description des mots système %SW30 à %SW47

Description détaillée

Description des mots système %SW30 à %SW35 :

Mot Symbole	Fonction	Description	Etat initial	Modicon M340	Premium	Quantum
%SW30 MASTCURRTIME	Temps d'exécution tâche maître	Ce mot indique le temps d'exécution du dernier cycle de la tâche maître (en ms).	-	OUI	OUI	OUI
%SW31 MASTMAXTIME	Temps d'exécution maxi tâche maître	Ce mot indique le temps d'exécution le plus long de la tâche maître depuis le dernier démarrage à froid (en ms).	-	OUI	OUI	OUI
%SW32 MASTMINTIME	Temps d'exécution mini tâche maître	Ce mot indique le temps d'exécution le plus court de la tâche maître depuis le dernier démarrage à froid (en ms).	-	OUI	OUI	OUI
%SW33 FASTCURRTIME	Temps d'exécution tâche rapide	Ce mot indique le temps d'exécution du dernier cycle de la tâche rapide (en ms). Il n'est pas disponible sur les automates de sécurité Quantum.	-	OUI	OUI	OUI (sauf pour les automates de sécurité)
%SW34 FASTMAXTIME	Temps d'exécution maxi tâche rapide	Ce mot indique le temps d'exécution le plus long de la tâche rapide depuis le dernier démarrage à froid (en ms). Il n'est pas disponible sur les automates de sécurité Quantum.	-	OUI	OUI	OUI (sauf pour les automates de sécurité)
%SW35 FASTMINTIME	Temps d'exécution mini tâche rapide	Ce mot indique le temps d'exécution le plus court de la tâche rapide depuis le dernier démarrage à froid (en ms). Il n'est pas disponible sur les automates de sécurité Quantum.	-	OUI	OUI	OUI (sauf pour les automates de sécurité)

NOTE : Le temps d'exécution est le temps écoulé entre le début (acquisition des entrées) et la fin (mise à jour des sorties) d'un cycle de scrutation. Ce temps inclut le traitement des tâches événementielles, la tâche rapide ainsi que le traitement des requêtes console.

Description des mots système %SW36 à %SW47 :

Mot **Symbole**	**Fonction**	**Description**	**Etat initial**	**Modicon M340**	**Quantum**	**Premium**
%SW36 AUX0CURRTIME **%SW39** AUX1CURRTIME **%SW42** AUX2CURRTIME **%SW45** AUX3CURRTIME	Temps d'exécution tâches auxiliaires	Ces mots indiquent le temps d'exécution du dernier cycle des tâches auxiliaires AUX 0... 3 (en ms). (1) uniquement sur les automates 140 CPU 6•• et TSX P57 5••. Ces mots ne sont pas disponibles sur les automates de sécurité Quantum.	-	NON	OUI (1)	OUI (1) (sauf pour les automates de sécurité)
%SW37 AUX0MAXTIME **%SW40** AUX1MAXTIME **%SW43** AUX2MAXTIME **%SW46** AUX3MAXTIME	Temps d'exécution maxi tâches auxiliaires	Ces mots indiquent le temps d'exécution le plus long des tâches auxiliaires AUX 0... 3 depuis le dernier démarrage à froid (en ms). (1) uniquement sur les automates 140 CPU 6•• et TSX P57 5••. Ces mots ne sont pas disponibles sur les automates de sécurité Quantum.	-	NON	OUI (1)	OUI (1) (sauf pour les automates de sécurité)
%SW38 AUX0MINTIME **%SW41** AUX1MINTIME **%SW44** AUX2MINTIME **%SW47** AUX3MINTIME	Temps d'exécution mini tâches auxiliaires	Ces mots indiquent le temps d'exécution le plus court des tâches auxiliaires AUX 0... 3 depuis le dernier démarrage à froid (en ms). (1) uniquement sur les automates 140 CPU 6•• et TSX P57 5••. Ces mots ne sont pas disponibles sur les automates de sécurité Quantum.	-	NON	OUI (1)	OUI (1) (sauf pour les automates de sécurité)

Description des mots système %SW48 à %SW59

Description détaillée

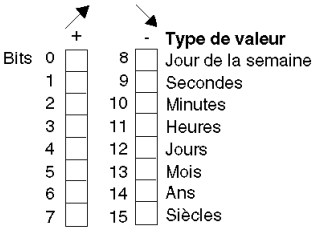
Description des mots système %SW48 à %SW59.

Mot Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum
%SW48 IOEVTNB	Nombre d'événements	Ce mot indique le nombre d'événements traités depuis le dernier démarrage à froid (en ms). Il peut être écrit par le programme ou par le terminal. Ce mot n'est pas disponible sur les automates de sécurité Quantum.	0	OUI	OUI	OUI (sauf pour les automates de sécurité)

%SW49
DAYOFWEEK
%SW50
SEC
%SW51
HOURMIN
%SW52
MONTHDAY
%SW53
YEAR
	Fonction d'horodateur	Mots système contenant la date et l'heure courantes (en BCD) :
		• %SW49 : jour de la semaine :
		  • 1 = lundi,
		  • 2 = mardi,
		  • 3 = mercredi,
		  • 4 = jeudi,
		  • 5 = vendredi,
		  • 6 = samedi,
		  • 7 = dimanche.
		• %SW50 : secondes (16#SS00)
		• %SW51 : heures et minutes (16#HHMM)
		• %SW52 : mois et jour (16#MMJJ)
		• %SW53 : année (16#AAAA)
		Ces mots sont gérés par le système lorsque le bit %S50 est paramétré sur 0.
		Ils sont écrits par le programme utilisateur ou par le terminal lorsque le bit %S50 est paramétré sur 1.
			-	OUI	OUI	OUI

Mot Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum

%SW54
STOPSEC
%SW55
STOPHM
%SW56
STOPMD
%SW57
STOPYEAR
%SW58
STOPDAY
	Fonction d'horodateur au dernier arrêt	Mots système contenant la date et l'heure de la dernière coupure du secteur ou du dernier arrêt de l'automate (au format Binary Coded Decimal) :
		• %SW54 : secondes (00SS),
		• %SW55 : heures et minutes (HHMM),
		• %SW56 : mois et jour (MMJJ),
		• %SW57 : année (AAAA).
		• %SW58 : l'octet de poids fort contient le jour de la semaine (de 1 pour lundi à 7 pour dimanche), tandis que l'octet de poids faible contient le code du dernier arrêt :
		  • 1 = passage du mode RUN au mode STOP par le biais du terminal ou de l'entrée dédiée,
		  • 2 = arrêt par le chien de garde (tâche de l'automate ou débordement SFC),
		  • 4 = coupure secteur ou opération de verrouillage de la carte mémoire,
		  • 5 = arrêt suite à une défaillance matérielle,
		  • 6 = arrêt suite à une défaillance logicielle. Les détails sur le type de défaillance logicielle sont stockés dans %SW125.
			-	OUI	OUI	OUI

Mot Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum
%SW59 ADJDATETIME	Réglage de la date courante	Contient deux séries de 8 bits permettant de régler la date courante. L'action est toujours effectuée sur le front montant du bit. Ce mot est activé par le bit %S59 = 1. Dans l'illustration ci-dessous, les bits de la colonne de gauche incrémentent la valeur, tandis que ceux de la colonne de droite la décrémentent : 	0	OUI	OUI	OUI

Description des mots système %SW70 à %SW100

Description détaillée

Description des mots système %SW70 à %SW100.

Mot Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum
%SW70 WEEKOFYEAR	Fonction d'horodateur	Mot système contenant le numéro qu'occupe la semaine dans l'année : 1 à 52 (en BCD).	-	OUI	OUI	OUI
%SW71 KEY_SWITCH	Position des commutateurs sur le panneau avant de l'automate Quantum	Ce mot fournit l'image des positions des commutateurs sur le panneau avant du processeur Quantum. Il est mis à jour automatiquement par le système. ● Commutateur %SW71.0 = 1 en position « Protection mémoire », ● commutateur %SW71.1 = 1 en position « STOP », ● commutateur %SW71.2 = 1 en position « START », ● commutateur %SW71.8 = 1 en position « MEM », ● commutateur %SW71.9 = 1 en position « ASCII », ● commutateur %SW71.10 = 1 en position « RTU », ● les commutateurs %SW71.3 à 7 et 11 à 15 ne sont pas utilisés.	0	NON	NON	OUI
%SW75 TIMEREVTNB	Compteur d'événements de type temporisateur	Ce mot contient le nombre d'événements de type temporisateur de la file d'attente. **(1)** : non disponible sur les processeurs suivants : TSX 57 1•/2•/3•/4•/5•. Ce mot n'est pas disponible sur les automates de sécurité Quantum.	0	OUI	OUI (1)	OUI (sauf pour les automates de sécurité)
%SW76 DLASTREG	Fonction de diagnostic : enregistrement	Résultat du dernier enregistrement ● = 0 si l'enregistrement est réussi ● = 1 si le tampon de diagnostic n'a pas été configuré ● = 2 si le tampon de diagnostic est plein	0	OUI	OUI	OUI

Mot Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum
%SW77 DLASTDEREG	Fonction de diagnostic : non-enregistrement	Résultat du dernier désenregistrement ● = 0 si le non-enregistrement est réussi ● = 1 si le tampon de diagnostic n'a pas été configuré ● = 21 si l'identificateur d'erreur n'est pas valide ● = 22 si l'erreur n'a pas été enregistrée	0	OUI	OUI	OUI
%SW78 DNBERRBUF	Fonction de diagnostic : nombre d'erreurs	Nombre d'erreurs actuellement dans le tampon de diagnostic.	0	OUI	OUI	OUI
%SW80 MSGCNT0 **%SW81** MSGCNT1	Gestion des messages	Ces mots sont mis à jour par le système et peuvent également être réinitialisés à l'aide de %S80. Pour Premium : ● %SW80 : nombre de messages envoyés par le système au port terminal (port Uni-Telway) ● %SW81 : nombre de messages reçus par le système au port terminal (port Uni-Telway) Pour Modicon M340 : ● %SW80 : nombre de messages envoyés par le système au port terminal (port série Modbus) ● %SW81 : nombre de messages reçus par le système au port terminal (port série Modbus) Pour Quantum : ● %SW80 : nombre de messages Modbus envoyés par le système comme client sur tous les ports de communication. **NOTE :** Les messages Modbus envoyés par le système comme maître ne sont pas comptés dans ce mot. ● %SW81 : nombre de messages Modbus envoyés par le système comme client sur tous les ports de communication. **NOTE :** Les messages Modbus reçus en réponse aux requêtes envoyées par le système, comme maître, ne sont pas comptés dans ce mot.	0	OUI	OUI	OUI

Mot **Symbole**	**Fonction**	**Description**	**Etat initial**	**Modicon M340**	**Premium Atrium**	**Quantum**
%SW82 **%SW83**	Gestion des messages	Ces mots sont mis à jour par le système et peuvent également être réinitialisés à l'aide de %S80. Pour Premium : ● %SW82 : nombre de messages envoyés par le système au module PCMCIA ● %SW83 : nombre de messages envoyés par le module PCMCIA et reçus par le système Pour Quantum : ● %SW82 : nombre de messages Modbus envoyés ou reçus sur le port série 1 ● %SW83 : nombre de messages Modbus envoyés ou reçus sur le port série 2	0	NON	OUI	OUI
%SW84 MSGCNT4 **%SW85** MSGCNT5	Premium : gestion Télégramme Modicon M340 : Gestion des messages	Ces mots sont mis à jour par le système et peuvent également être réinitialisés à l'aide de %S80. Pour Premium : ● %SW84 : nombre de télégrammes envoyés par le système ● %SW85 : nombre de télégrammes reçus par le système Pour Modicon M340 : ● %SW84 : nombre de messages envoyés au port USB ; ● %SW85 : nombre de messages reçus par le port USB.	0	OUI	OUI	NON
%SW86 MSGCNT6	Gestion des messages	Ce mot est mis à jour par le système et peut également être réinitialisé à l'aide de %S80. Pour Premium : ● Nombre de messages refusés par le système. Pour Modicon M340 : ● Nombre de messages refusés par le système et non traités en raison du manque de ressources, par exemple. Si le message est refusé par le serveur Modbus, il correspond alors aux messages d'exception Modbus envoyés par l'UC au client Modbus distant.	0	OUI	OUI	NON

Mot Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum
%SW87 MSTSERVCNT	gestion des communications	Nombre de requêtes traitées par le serveur synchrone par cycle de tâche maître (MAST). Les requêtes traitées peuvent provenir de tous les ports de communication (ayant accès au serveur Modbus/UNI-TE, chacun avec sa propre limitation). Cela signifie également que les requêtes d'autres clients, et par conséquent les fonctions élémentaires de communication comme le scrutateur d'ES, un module IHM connecté, etc., sont prises en compte.	0	OUI	OUI	OUI
%SW88 ASNSERVCNT **%SW89** APPSERVCNT	Premium : gestion des communications Modicon M340 : requêtes HTTP et FTP reçues, par seconde, par les serveurs Web et FTP du processeur	Pour Premium : ● %SW88 : nombre de requêtes traitées par le serveur asynchrone par cycle de tâche maître (MAST) ● %SW89 : nombre de requêtes traitées par les fonctions de serveur (immédiatement) par cycle de tâche maître (MAST) Pour Modicon M340 : ● %SW88 : nombre de requêtes HTTP reçues, par seconde, par le serveur Web du processeur ● %SW89 : nombre de requêtes FTP reçues, par seconde, par le serveur FTP	0	OUI	OUI	NON

Mot Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum
%SW90 MAXREQNB	Nombre maximum de requêtes traitées par cycle de tâche maître	Ce mot est utilisé pour définir un nombre maximum de requêtes (tous protocoles inclus : UNI-TE, Modbus, etc.) qui peuvent être traitées par l'automate par cycle de tâche maître. (Les requêtes envoyées par l'automate en tant que client ne sont pas concernées.) Ce nombre de requêtes doit être compris entre un minimum et un maximum (défini comme N+4) en fonction du modèle. **Pour la gamme M340 :** ● BMX P34 10••/20••/ : N = 8 (minimum 2, maximum 8 + 4 = 12), **Pour la gamme Premium :** ● TSX 57 0• : N = 4 (minimum 2, maximum 4 + 4 = 9), ● TSX 57 1• : N = 4 (minimum 2, maximum 4 + 4 = 8), ● TSX 57 2• : N = 8 (minimum 2, maximum 8 + 4 = 12), ● TSX 57 3• : N = 12 (minimum 2, maximum 12 + 4 = 16), ● TSX 57 4• : N = 16 (minimum 2, maximum 16 + 4 = 20), ● TSX 57 5• : N = 16 (minimum 2, maximum 16 + 4 = 20), **Pour la gamme Quantum :** ● 140 CPU 31••/43••/53••/ : N = 10 (minimum 5, maximum 10 + 4 = 14), ● 140 CPU 6•• : N = 20 (minimum 5, maximum 20 + 4 = 24),	N	OUI	OUI	OUI

Mot **Symbole**	**Fonction**	**Description**	**Etat initial**	**Modicon M340**	**Premium Atrium**	**Quantum**
Suite **%SW90** MAXREQNB	Nombre maximum de requêtes traitées par cycle de tâche maître	Le mot est initialisé par le système avec la valeur N (valeur par défaut). Si la valeur 0 est saisie, la valeur N est prise en compte. Si une valeur comprise entre 1 et le minimum est saisie, la valeur minimum est prise en compte. Si une valeur supérieure au maximum est saisie, la valeur maximum est prise en compte. Le nombre de requêtes à traiter par cycle doit prendre en compte les requêtes de tous les ports de communication (ayant accès au serveur). Cela signifie que les requêtes d'autres clients que les fonctions élémentaires de communication, comme le scrutateur d'ES, un module IHM connecté, etc., sont prises également en compte.	N	OUI	OUI	OUI
%SW91-92	Vitesses de messages de blocs fonction	• `%SW91` : nombre de messages de blocs fonction envoyés par seconde • `%SW92` : nombre de messages de blocs fonction reçus par seconde Peut être lu par le programme utilisateur ou par le terminal. Ces compteurs n'incluent pas d'autres requêtes sortantes provenant d'un scrutateur d'ES, par exemple.	0	OUI	OUI	NON

Mot Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum
%SW93	Commande et état du format du système de fichiers de carte mémoire	Peut être lu et écrit par le programme utilisateur ou par le terminal. Ce mot est utilisé par le client pour formater ou nettoyer la carte mémoire : L'opération de formatage supprime les pages Web. Pour les rétablir, effectuez l'une des actions suivantes : ● Utilisez le FTP. ● Avant d'effectuer le formatage, sauvegardez les pages Web en les copiant par FTP. ● Après le formatage, rechargez les pages Web par FTP. ● Réinstallez le micrologiciel de système d'exploitation du processeur. L'opération de nettoyage efface le contenu du répertoire de stockage de données. Le formatage et le nettoyage ne sont possibles qu'en mode Stop : ● `%SW93.0 = 1` un front montant démarre l'opération de formatage. ● `%SW93.1` affiche l'état du système de fichiers après une requête de formatage ou de nettoyage : ● `%SW93.1 = 0` système de fichier incorrect ou commande en cours. ● `%SW93.1 = 1` système de fichiers valide. ● `%SW93.0 = 1` un front montant démarre l'opération de nettoyage.	0	OUI	NON	NON
%SW94 **%SW95**	Signature de modification de l'application	Ces deux mots contiennent une valeur 32 bits qui change à chaque modification de l'application sauf lors : ● d'une mise à jour des informations de chargement, ● d'un remplacement de la valeur initiale par la valeur courante, ● d'un enregistrement de la commande de paramètre. Ils peuvent être lus par le programme utilisateur ou par le terminal.	-	OUI	NON	NON

Mot Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum

%SW96
CMDDIAGSA-VEREST
	Commande et diagnostic d'enregistrement et de restauration	Ce mot est utilisé pour copier la valeur courante de %MW dans la mémoire Flash interne *(voir page 107)* ou pour la supprimer de cette même mémoire, et pour fournir l'état de l'action. Il peut être lu par le programme utilisateur ou par le terminal :
		• %SW96.0 : requête de copie de la valeur courante de %MW dans la mémoire Flash interne. Paramétrée sur 1 par l'utilisateur pour demander un enregistrement et paramétrée sur 0 par le système lorsqu'un enregistrement est en cours.
		NOTE : vous devez arrêter le processeur avant d'effectuer la copie à l'aide de %SW96.0.
		• %SW96.1 est réglé sur 1 par le système lorsqu'un enregistrement est terminé et sur 0 par le système lorsqu'un enregistrement est en cours.
		• %SW96.2 = 1 indique une erreur lors d'une opération d'enregistrement ou de restauration (voir %SW96.8 à 15 pour les définitions de codes d'erreur).
		• %SW96.3 = 1 indique qu'une opération de restauration est en cours.
		• %SW96.4 peut être réglé sur 1 par l'utilisateur pour supprimer la zone %MW de la mémoire Flash interne.
		• %SW96.7 = 1 indique que la mémoire interne contient un enregistrement %MW valide.
		• %SW96.8 à 15 sont des codes d'erreur utilisés lorsque %SW96.2 est réglé sur 1 :
		  • %SW96.9 = 1 indique que le nombre %MW enregistré est inférieur au nombre configuré.
		  • %SW96.8 = 1 et %SW96.9 = 1 signifient que le nombre %MW enregistré est supérieur au nombre configuré,
		  • %SW96.8 = 1, %SW96.9 = 1 et %SW96.10 = 1 indiquent une erreur d'écriture dans la mémoire Flash interne.
			-	OUI	NON	NON

Mot Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum
%SW97 CARDSTS	Etat de la carte	Peut être lu par le programme utilisateur ou par le terminal. Indique l'état de la carte. %SW97 : 0000 = pas d'erreur. 0001 = sauvegarde de l'application ou écriture dans un fichier envoyée sur une carte protégée en écriture. 0002 = carte non reconnue ou endommagement de la sauvegarde de l'application. 0003 = sauvegarde de l'application requise, mais aucune carte disponible. 0004 = erreur d'accès à la carte, par exemple après la suppression incorrecte d'une carte. 0005 = aucun système de fichiers présent sur la carte ou système présent non compatible. Utilisez %SW93.0 pour formater la carte.	-	OUI	NON	NON
%SW99 [1] INPUTADR/ SWAP [1]	Gestion de redondance des communications(1)	**NOTE :** ce mot est utilisé pour les modules Premium et Quantum, mais avec des fonctions différentes. Mot utilisé pour prendre en charge la redondance des modules réseau. Lorsqu'un problème est détecté sur un module de communication utilisé pour accéder à un numéro de réseau x (X-WAY), il est possible de basculer vers un autre module de communication (connecté au même réseau) en saisissant le numéro de réseau dans le mot %SW99. %SW99 est réinitialisé à 0 par le système.	0	NON	OUI(1)	NON

Mot Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum
%SW99 [2] CRA_COMPAT_HIGH [2]	Registre d'état de compatibilité élevée CRA	**NOTE :** ce mot est utilisé pour les modules Premium et Quantum, mais avec des fonctions différentes. Le mot est incrémenté à chaque fois qu'une modification CCOTF est effectuée sur un automate. Lorsqu'un module est inséré dans la station d'E/S distantes, le bit correspondant est réglé sur 1 et indique que le module est connecté à la station et compatible CCOTF.	0	NON	NON	OUI(2)
%SW100 CCOTF_COUNT	Registre d'état de comptage CCOTF	Mot utilisé pour gérer la compatibilité CCOTF lorsqu'un nouveau module est inséré. %SW100 = XXYY où : ● XX est incrémenté chaque fois qu'une configuration d'E/S est effectuée en mode RUN sur une station d'E/S distantes, ● YY est incrémenté chaque fois qu'une configuration d'E/S est effectuée en mode RUN sur un rack local.	0	NON	NON	OUI

Description des mots système %SW108 à %SW116

Description détaillée

Description du mot système %SW108 à %SW116.

Mot Symbole	Fonction	Description	Etat initial	Modicon M340	Quantum	Premium Atrium
%SW108 FORCEDIOIM	Nombre de bits de modules d'E/S forcés	Ce mot système compte le nombre de bits de module d'entrées/sorties forcés. Ce mot est incrémenté lors d'un forçage ou décrémenté, lors d'un déforçage.	0	OUI	OUI	OUI
%SW109 FORCEDANA	Nombre de voies analogiques forcées	Ce mot système compte le nombre de voies analogiques forcés. Ce mot est incrémenté lors d'un forçage ou décrémenté, lors d'un déforçage.	0	OUI	NON	OUI
%SW116 REMIOERR	Défaut d'E/S Fipio	Normalement à 0, chaque bit de ce mot est significatif d'un statut d'échange Fipio dans laquelle il est testé. Ce mot est remis à 0 par l'utilisateur. Détail des bits du mot %SW116 : ● %SW116.0 = 1 erreur d'échange explicite (la variable n'est pas échangée sur le bus) ● %SW116.1 = 1 temporisation sur un échange explicite (pas de réponse au bout de la temporisation) ● %SW116.2 = 1 nombre maximum d'échanges explicites simultanés atteint ● %SW116.3 = 1 une trame n'est pas correcte ● %SW116.4 = 1 la longueur d'une trame reçue supérieure à la longueur déclarée ● %SW116.5 = réservé à 0 ● %SW116.6 = 1 une trame est invalide, ou un agent s'initialise ● %SW116.7 = 1 absence d'un équipement configuré ● %SW116.8 = 1 défaut voie (au moins une voie d'un équipement signale un défaut) ● %SW116.9 à 15 = réservé à 0	-	NON	NON	OUI

Description des mots système %SW123 à %SW127

Description détaillée

Description des mots système %SW123 à %SW127.

Mot Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum
%SW123 ADJBUSX	Autorisation système pour le BUS X	Ce mot système est réservé au système et ne peut pas être utilisé par l'application utilisateur.	-	OUI	OUI	NON
%SW124 CPUERR	Type d'erreur du processeur ou du système	Le dernier type de défaut système rencontré est écrit dans ce mot par le système (ces codes restent inchangés lors d'un redémarrage à froid) : ● 16#30: défaut de codage système, ● 16#53: erreur de timeout lors d'échanges d'E/S, ● 16#60 à 64 : débordement de pile, ● 16#65: La durée d'exécution de tâche rapide est trop basse ● 16#81: détection d'une erreur d'embase **REMARQUE :** le code système 16#81 n'est pas pris en charge par les automates Quantum. **REMARQUE :** si cette erreur est détectée, tous les racks doivent être réinitialisés. ● 16#90: défaut d'interrupteur système : problème informatique imprévu.	-	OUI	OUI	OUI

Mot Symbole	Fonction	Description	Etat initial	Modicon M340	Premium Atrium	Quantum
%SW125 BLKERRTYPE	Dernier défaut détecté	Le code du dernier défaut détecté est donné dans ce mot. Les codes d'erreur suivants entraînent l'arrêt de l'automate si %S78 est paramétré sur 1. %S15, %S18 et %S20 sont toujours actifs indépendamment de %S78 : ● 16#2258: exécution de l'instruction HALT, ● 16#DE87 : erreur de calcul sur les nombres à virgule flottante (%S18, ces erreurs sont répertoriées dans le mot %SW17), ● 16#DEB0 : débordement du chien de garde (%S11), ● 16#DEF0 : division par 0 (%S18), ● 16#DEF1 : erreur de transfert de chaîne de caractères (%S15), ● 16#DEF2 : erreur arithmétique ; %S18, ● 16#DEF3 : dépassement d'index (%S20). **NOTE :** les codes 16#8xF4, 16#9xF4 et 16#DEF7 indiquent une erreur dans le diagramme fonctionnel en séquence (SFC).	-	OUI	OUI	OUI
%SW126 ERRADDR0 **%SW127** ERRADDR1	Adresse d'instruction de défaut bloquant	Adresse de l'instruction qui a généré le défaut bloquant de l'application. Pour les processeurs 16 bits, TSX P57 1••/2•• : ● %SW126 contient le décalage pour cette adresse. ● %SW127 contient le numéro du segment pour cette adresse. Pour les processeurs 32 bits : ● %SW126 contient le mot de poids faible pour cette adresse. ● %SW127 contient le mot de poids fort pour cette adresse.	0	OUI	OUI	OUI

6.3 Mots système spécifiques aux modules Atrium/Premium

Objet de cette section

Ce sous-chapitre décrit les mots système %SW128 à %SW167 dans le cas d'automates Premium et Atrium

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Description des mots système %SW60 à %SW65

Description détaillée

Description des mots système %SW60 à %SW65 sur la redondance d'UC Premium/Atrium.

Mot Symbole	Fonction	Description	Etat initial	Premium	Atrium

%SW60
HSB_CMD
	Registre de commande du système de redondance Premium	Signification des différents bits du mot %SW60 :
		• %SW60.1
		  • =0 configure l'automate A en mode OFFLINE.
		  • =1 configure l'automate A en mode RUN.
		• %SW60.2
		  • =0 configure l'automate B en mode OFFLINE.
		  • =1 configure l'automate B en mode RUN.
		• Différence de version au niveau du système d'exploitation %SW60.4
		  • =0 En cas de différence de version au niveau du système d'exploitation avec l'automate primaire, le mode Redondant passe en mode Hors ligne.
		  • =1 En cas de différence de version au niveau du système d'exploitation avec l'automate primaire, le mode Redondant reste en mode Redondant.
		Différence au niveau du système d'exploitation du micrologiciel. Ceci est lié à la version au niveau du système d'exploitation du processeur principal, la version au niveau du système d'exploitation du coprocesseur intégré, la version au niveau du système d'exploitation ETY surveillé et active un système de redondance d'UC pour permettre le fonctionnement avec des versions différentes au niveau du système d'exploitation s'exécutant sur le mode Primaire ou Redondant.
			0	OUI	NON

Mot Symbole	Fonction	Description	Etat initial	Premium	Atrium
%SW61 HSB_STS	Registre d'état du système de redondance Premium	Signification des différents bits du mot %SW61.0 à %SW61.6 : ● %SW61.0 et %SW61.1 Etat de l'automate local. ● %SW61.1=0 et %SW61.0=1 : mode OFFLINE. ● %SW61.1=1 et %SW61.0=0 : mode primaire. ● %SW61.1=1 et %SW61.0=1 : mode de redondance de l'UC. ● %SW61.2 et %SW61.3 Etat de l'automate distant. ● %SW61.3=0 et %SW61.2=1 : mode OFFLINE. ● %SW61.3=1 et %SW61.2=0 : mode primaire. ● %SW61.3=1 et %SW61.2=1 : mode de redondance de l'UC : ● %SW61.3=0 et %SW61.2=0 : l'automate distant n'est pas accessible (hors tension, aucune communication). ● %SW61.4 =1 : en cas de détection d'une différence de logique entre les automates primaire et redondant. ● %SW61.5 = 0 ou 1, en fonction de l'adresse MAC du coprocesseur Ethernet : ● =0 l'automate avec la plus petite adresse MAC devient l'automate A. ● =1 l'automate avec la plus grande adresse MAC devient l'automate B. ● %SW61.6 : ce bit indique si la liaison sync UC entre deux automates est valide : ● %SW61.6=0: la liaison sync UC est valide. Le contenu du bit 5 est significatif. ● %SW61.6=1: la liaison sync UC n'est pas valide. Dans ce cas, le contenu du bit 5 n'est pas significatif parce que la comparaison de deux adresses MAC ne peut pas être effectuée.	0	OUI	NON

Mot Symbole	Fonction	Description	Etat initial	Premium	Atrium

%SW61
HSB_STS
	Registre d'état du système de redondance Premium	Signification des différents bits du mot %SW61.7 à %SW61.9 :
		● %SW61.7 : ce bit indique s'il y a une différence de version au niveau du système d'exploitation du processeur principal entre les modes Primaire et Redondant :
		  ● =0: Aucune différence de version au niveau du système d'exploitation du micrologiciel.
		  ● =1: Différence de version au niveau du système d'exploitation. Si la différence de version au niveau du système d'exploitation n'est pas autorisée dans le registre de commande (bit 4 = 0), le système ne fonctionnera pas en mode redondant tant que le défaut est signalé.
		● %SW61.8 : ce bit indique s'il y a une différence de version au niveau du système d'exploitation du coprocesseur entre les modes Primaire et Redondant :
		  ● =0: Aucune différence de version au niveau du système d'exploitation du coprocesseur.
		  ● =1: Différence de version au niveau du système d'exploitation du coprocesseur. Si la différence de version au niveau du système d'exploitation n'est pas autorisée dans le registre de commande (bit 4 = 0), le système ne fonctionnera pas en mode redondant tant que le défaut est signalé.
		● %SW61.9 : ce bit indique si au moins un module ETY n'est pas conforme à la version minimum :
		  ● =0: tous les modules ETY sont conformes à la version minimum.
		  ● =1: Au moins un module ETY n'est pas conforme à la version minimum. Dans ce cas, aucun automate primaire ne peut démarrer.
			0	OUI	NON

Mot Symbole	Fonction	Description	Etat initial	Premium	Atrium

%SW61
HSB_STS
	Registre d'état du système de redondance Premium	Signification des différents bits du mot %SW61.10 et %SW61.15 :
		● %SW61.10 : ce bit indique s'il y a une différence de version au niveau du système d'exploitation ETY surveillé entre les modes Primaire et Redondant :
		  ● =0: Aucune différence de version au niveau du système d'exploitation des modules ETY surveillés.
		  ● =1: Différence de version au niveau du système d'exploitation des modules ETY surveillés. Si la différence de version au niveau du système d'exploitation n'est pas autorisée dans le registre de commande (bit 4 = 0), le système ne fonctionnera pas en mode redondant tant que le défaut est signalé.
		● %SW61.15 : Si %SW61.15 est réglé sur 1, cela signifie que le coprocesseur Ethernet est correctement configuré et qu'il fonctionne.
			0	OUI	NON

%SW62 HSBY_REVERSE0 **%SW63** HSBY_REVERSE1 **%SW64** HSBY_REVERSE2 **%SW65** HSBY_REVERSE3	Mot de transfert Premium	Ces quatre mots sont des registres inversés réservés au processus de transfert inversé. Ces quatre registres inversés peuvent être écrits dans le programme d'application (première section) de l'automate redondant, puis sont transférés lors de chaque cycle vers l'automate primaire.	0	OUI	NON

Description des mots système %SW128 à %SW143

Description détaillée

Description des mots système %SW128 à SW143 :

Mot **Symbole**	**Fonction**	**Description**	**Etat initial**
%SW128...143 ERRORCNXi avec i=0 à 15	Point de connexion Fipio en défaut	Chaque bit de ce groupe de mots est significatif de l'état d'un équipement connecté sur le bus Fipio. Normalement à 1, la présence à 0 d'un de ces bits indique l'apparition d'un défaut sur ce point de connexion. Pour un point de connexion non configuré, le bit correspondant vaut toujours 1.	0

Tableau de correspondance entre les bits des mots et l'adresse d'un point de connexion :

	Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7	Bit 8	Bit 9	Bit 10	Bit 11	Bit 12	Bit 13	Bit 14	Bit 15
%SW128	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
%SW129	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
%SW130	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
%SW131	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
%SW132	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
%SW133	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
%SW134	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
%SW135	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
%SW136	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
%SW137	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
%SW138	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
%SW139	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
%SW140	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
%SW141	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
%SW142	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
%SW143	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

Description des mots système %SW144 à %SW146

Description détaillée

Description des mots système %SW144 à %SW146.

Mot Symbole	Fonction	Description	Etat initial

%SW144
BAOPMOD
	Mode de marche fonction arbitre de bus Fipio	Ce mot système permet l'arrêt et le démarrage de la fonction arbitre de bus et de la fonction producteur/consommateur. Il permet de modifier le mode de démarrage, le mode automatique et le mode manuel du bus en cas d'arrêt.
		• %SW144.0
		  • = 1 : fonction producteur/consommateur en RUN.
		  • = 0 : fonction producteur/consommateur en STOP (aucune variable n'est échangée sur le bus).
		• %SW144.1
		  • = 1 : l'arbitre de bus est en RUN 0.
		  • = 0 : l'arbitre de bus est en STOP (aucune scrutation de variables et de messages n'est réalisée sur le bus).
		• %SW144.2
		  • = 1 : démarrage automatique en cas d'arrêt automatique du bus.
		  • = 0 : démarrage manuel en cas d'arrêt automatique du bus.
		• %SW144.3 à 15 réservés, %SW144.3 = 1, %SW144.4 à 15 = 0.
			0

%SW145 BAPARAM	Modification des paramètres de l'arbitre de bus Fipio	Les bits sont réglés sur 1 par l'utilisateur et remis à 0 par le système lorsque l'initialisation a été effectuée. • %SW145.0 = 1 : modification de la priorité de l'arbitre de bus ; l'octet de poids fort de ce mot système contient la valeur de la priorité de l'arbitre de bus qui sera appliquée au bus. • %SW145.1 et %SW145.2 sont réservés. • %SW145.3 à %SW145.7 réservés à 0. • %SW145.8 à %SW145.15 : cet octet contient la valeur qui est appliquée au bus, suivant la valeur du bit 0. La modification de ces paramètres peut se faire lorsque l'arbitre de bus est en RUN, mais leur prise en compte par l'application nécessite un arrêt puis un redémarrage de celle-ci.	0

Mot Symbole	Fonction	Description	Etat initial
%SW146 BASTATUS	Fonction d'arbitre sur bus Fipio	L'octet de poids faible indique l'état de la fonction producteur/consommateur. L'octet de poids fort indique l'état de la fonction arbitre de bus. Valeur de l'octet : ● 16#00 : la fonction n'existe pas (pas d'application Fipio). ● 16#70 : la fonction est initialisée mais pas opérationnelle (en STOP). ● 16#F0 : la fonction est en cours d'exécution normale (en RUN).	0

⚠ ATTENTION

FONCTIONNEMENT SYSTEME INATTENDU

La modification des mots système %SW144 et %SW145 peut entraîner l'arrêt de l'automate.

Le non-respect de ces instructions peut provoquer des blessures ou des dommages matériels.

Description des mots système %SW147 à %SW152

Description détaillée

Description des mots système %SW147 à %SW152 :

Mot **Symbole**	**Fonction**	**Description**	**Etat initial**
%SW147 TCRMAST	Temps de cycle réseau MAST	Une valeur non nulle indique (en ms) la valeur du temps de cycle réseau (TCRMAST) de la tâche MAST.	0
%SW148 TCRFAST	Temps de cycle réseau FAST	Une valeur non nulle indique (en ms) la valeur du temps de cycle réseau (TCRFAST) de la première tâche FAST.	0
%SW150 NBFRSENT	Nombre de trames émises	Ce mot indique le nombre de trames émises par le gestionnaire de la voie Fipio.	0
%SW151 NBFRREC	Nombre de trames reçues	Ce mot indique le nombre de trames reçues par le gestionnaire de la voie Fipio.	0
%SW152 NBRESENTMSG	Nombre de messages repris	Ce mot indique le nombre de reprises de messages effectués par le gestionnaire de la voie Fipio.	0

Description des mots système %SW153

Description détaillée

Description du mot système %SW153 :

Mot Symbole	Fonction	Description	Etat initial
%SW153 FipioERR0	Liste des défauts du gestionnaire de la voie Fipio	Chaque bit est mis à 1 par le système et remis à 0 par l'utilisateur. Voir liste ci-dessous.	0

Description des bits

- bit 0 = "défaut d'overrun de la station" : correspond à une perte de symbole MAC en réception, liée à une réaction trop lente du récepteur.
- bit 1 = " défaut de refus message" : indique un message avec acquittement refusé ou sans acquittement MAC en réception,
- bit 2 = "défaut de refus de variable d'interruption".
- bit 3 = "défaut d'underrun de la station" : correspond à une incapacité de la station à respecter la vitesse d'émission sur le réseau.
- bit 4 = "défaut de couche physique" : correspond à une absence prolongée de transmission au niveau couche physique.
- bit 5 = "défaut de non écho" : correspond à un défaut pour lequel l'émetteur est en cours d'émission, avec un courant d'émission compris dans la plage de fonctionnement, et simultanément détection d'absence de signal sur la même voie.
- bit 6 = "défaut de bavardage" : correspond à un défaut pour lequel l'émetteur dispose du contrôle de la ligne depuis un temps supérieur à la limite maximale de fonctionnement définie. Ce défaut est par exemple provoqué par une détérioration du modulateur ou par une couche liaison de données défectueuse.
- bit 7 = "défaut d'hypocourant" : correspond à un défaut pour lequel l'émetteur produit sur la ligne, lorsqu'il est sollicité, un courant inférieur à la limite minimale de fonctionnement définie. Ce défaut est provoqué par une élévation de l'impédance de ligne (ligne ouverte, etc.).
- bit 8 = "défaut de trame perforée" : indique la réception d'un silence dans le corps d'une trame après l'identification d'un délimiteur de début de trame et avant l'identification d'un délimiteur de fin de trame. L'apparition d'un silence dans des conditions normales de fonctionnement a lieu après l'identification d'un délimiteur de fin de trame.
- bit 9 = "défaut de CRC trame en réception" : indique une différence de valeur entre le CRC calculé sur la trame normalement reçue et le CRC contenu dans cette trame.

- bit 10 = "défaut de codage trame en réception" : indique la réception de certains symboles, appartenant exclusivement aux séquences de délimitation du début et de la fin de trame, dans le corps d'une trame.
- bit 11 = "défaut de longueur de la trame reçue" : le nombre d'octets reçus pour le corps d'une trame est supérieur à 256 octets.
- bit 12 = "réception d'une trame de type inconnu" : dans le corps d'une trame, le premier octet identifie le type de trame liaison. Un certain nombre de types de trames sont définis dans le protocole liaison de la norme WorldFip. L'existence de tout autre code dans une trame correspond à un défaut de type trame inconnue.
- bit 13 = "réception d'une trame tronquée" : un fragment de trame se caractérise par la reconnaissance d'une séquence de symboles du délimiteur de fin de trame alors que la station destinataire s'attend à recevoir un délimiteur de début de trame.
- bit 14 = "inutilisé, valeur non significative".
- bit 15 = "inutilisé, valeur non significative".

Description du mot système %SW154

Description détaillée

Description du mot système %SW154 :

Mot Symbole	Fonction	Description	Etat initial
%SW154 FipioERR1	Liste des défauts du gestionnaire de la voie Fipio	Chaque bit est mis à 1 par le système et remis à 0 par l'utilisateur. Voir liste ci-dessous.	0

Description des bits

- bit 0 = "time out séquence apériodique" : indique un dépassement de la fenêtre de messages ou de variables apériodiques dans un cycle élémentaire du macro-cycle.
- bit 1 = "refus de demande de messagerie" : indique une saturation de la file d'attente des messages, l'arbitre de bus n'est momentanément plus en mesure de mémoriser puis de satisfaire une demande.
- bit 2 = "refus de commande d'update urgente" : indique une saturation de la file d'attente des demandes d'échanges de variables apériodiques urgentes, l'arbitre de bus n'est momentanément plus en mesure de mémoriser ni de satisfaire la demande.
- bit 3 = "refus de commande d'update non urgente" : indique une saturation de la file d'attente des demandes d'échanges de variables apériodiques non urgentes, l'arbitre de bus n'est momentanément plus en mesure de mémoriser ni de satisfaire la demande.
- bit 4 = "défaut de silence" : l'arbitre de bus n'a détecté aucune activité sur le bus pendant une durée supérieure à un temps normalisé WorldFip.
- bit 5 = "collision sur le réseau à l'émission d'identificateur" : indique une activité sur le réseau pendant des périodes théoriques de silence. Entre une émission et l'attente d'une réponse par l'arbitre de bus, rien ne doit circuler sur le bus. Si l'arbitre de bus détecte une activité il génère un défaut de collision (Par exemple lorsque plusieurs arbitres sont actifs simultanément sur le bus).
- bit 6 = "défaut d'overrun de l'arbitre de bus" : indique un conflit d'accès à la mémoire de la station arbitre de bus.
- bit 7 = "inutilisé, valeur non significative".
- bit 8 à bit 15 = réservé à 0.

Description des mots système Premium/Atrium %SW155 à %SW167

Description détaillée

Description des mots système %SW155 à %SW167 :

Mot Symbole	Fonction	Description	Etat initial
%SW155 NBEXPLFIP	Nombre d'échanges explicites sur Fipio	Nombre d'échanges explicites en cours de traitement sur Fipio, effectués par instructions (READ_STS, REA_PARAM...). Prends en compte aussi les échanges explicites effectués par requêtes (READ_IO_OBJECT, WRITE_IO_OBJECT...) **Remarque :** le nombre d'échanges explicites est toujours inférieur à 24.	0
%SW160 à %SW167 PREMRACK0 à PREMRACK7	Etat de fonctionnement des modules automate	Les mots %SW160 à %SW167 sont associés respectivement aux racks 0 à 7. Les bits de 0 à 15 de chacun de ces mots sont associés aux modules situés aux positions 0 à 15 de ces racks. Le bit est à 0 si le module est en défaut, il est à 1 si le module fonctionne correctement. **Exemple :** %SW163.5 =0 Le module situé dans l'emplacement 5 du rack 3 est en défaut.	0

6.4 mots système spécifiques à Quantum

Objet de cette section

Cette section décrit les mots système %SW60 à %SW640 pour les automates Quantum.

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Description des mots système Quantum %SW60 à %SW65

Description détaillée

Description des mots système %SW60 à %SW65

Mot Symbole	Fonction	Description	Etat initial

%SW60
HSB_
CMD
	Registre de commande du système de redondance d'UC Quantum	Signification des différents bits du mot %SW60 :
		• %SW60.0=1 invalide les commandes saisies à l'écran (clavier).
		• %SW60.1
		  • =0 configure l'automate A en mode Hors ligne.
		  • =1 configure l'automate A en mode En ligne.
		• %SW60.2
		  • =0 configure l'automate B en mode Hors ligne.
		  • =1 configure l'automate B en mode En ligne.
		NOTE : l'automate d'UC primaire ne passe en mode RUN Local que si l'UC secondaire est en mode RUN Redondant.
		Au démarrage de l'automate secondaire, l'UC secondaire passe en mode Connecté (RUN Redondant) uniquement si les deux bits %SW60.1 et %SW60.2 sont réglés sur 1 (quel que soit l'affectation A/B).
		Si les bits %SW60.1 et %SW60.2 sont réglés simultanément sur 0, un basculement se produit :
		• le variateur primaire est exécuté en mode Run local, et
		• le variateur redondant est désormais le variateur primaire.
		Pour achever le basculement, les bits %SW60.1 et %SW60.2 doivent être remis à 1. L'UC opérant en mode Local revient en mode Connecté (RUN Redondant).
		Le mode Local/Connecté défini par les bits %SW60.1 et %SW60.2 n'est pas lié au mode Local/Connecté du clavier LCD.
		• %SW60.3=0 active le mode Hors ligne sur l'automate redondant si les applications sont différentes.
		• %SW60.4
		  • =0 n'autorise une mise à jour du micrologiciel qu'après l'arrêt de l'application.
		  • =1 autorise une mise à jour du micrologiciel même si l'application n'est pas arrêtée.
		• %SW60.5=1 demande de transfert d'application de l'automate redondant à l'automate primaire.
		• %SW60.8
		  • =0 commutation d'adresse sur le port Modbus 1 lors d'une permutation principale.
		  • =1 pas de commutation d'adresse sur le port Modbus 1 lors d'une permutation principale.
			0

Mot Symbole	Fonction	Description	Etat initial

		• %SW60.9
		  • =0 commutation d'adresse sur le port Modbus 2 lors d'une permutation principale.
		  • =1 pas de commutation d'adresse sur le port Modbus 2 lors d'une permutation principale.
		• %SW60.10
		  • =0 commutation d'adresse sur le port Modbus 3 lors d'une permutation principale.
		  • =1 pas de commutation d'adresse sur le port Modbus 3 lors d'une permutation principale.	

%SW61
HSB_
STS
	Registre d'état Quantum	Signification des différents bits du mot %SW61 :
		• Bits %SW61.0 et %SW61.1 du mode de marche de l'automate
		  • %SW61.1=0, %SW61.0=1 : mode Hors ligne.
		  • %SW61.1=1, %SW61.0=0 : mode primaire.
		  • %SW61.1=1, %SW61.0=1 : mode secondaire (redondant).
		• Bits %SW61.2 et %SW61.3 du mode de marche de l'autre automate
		  • %SW61.3=0, %SW61.2=1 : mode Hors ligne.
		  • %SW61.3=1, %SW61.2=0 : mode primaire.
		  • %SW61.3=1, %SW61.2=1 : mode secondaire (redondant).
		  • %SW61.3=0, %SW61.2=0 : l'automate distant n'est pas accessible (hors tension, aucune communication).
		• %SW61.4=0 les applications sont identiques sur les deux automates.
		• %SW61.5
		  • =0 l'automate est utilisé comme unité A.
		  • =1 l'automate est utilisé comme unité B.
		• %SW61.6 indique si la liaison sync UC entre deux automates est valide
		  • = 0 La liaison sync UC fonctionne correctement. Le contenu du bit 5 est significatif.
		  • = 1 la liaison sync UC n'est pas valide. Dans ce cas, le contenu du bit 5 n'est pas significatif parce que la comparaison de deux adresses MAC ne peut pas être effectuée.
		• %SW61.7
		  • =0 Version du SE des automates identique
		  • =1 Version du SE des automates différente.
		• %SW61.8
		  • =0 Version du SE des coprocesseurs identique.
		  • =1 Version du SE des coprocesseurs différente.
		• %SW61.12
		  • =0 Information donnée par le bit 13 non pertinente.
		  • =1 Information donnée par le bit 13 valide.
		  •
			0

Mot Symbole	Fonction	Description	Etat initial

		• %SW61.13
		  • =0 Adresse du module NOE définie comme égale à l'adresse IP.
		  • =1 Adresse du module NOE réglée sur adresse IP + 1.
		• %SW61.15
		  • =0 redondance d'UC non activée.
		  • =1 redondance d'UC activée.	

%SW62 HSBY_REVERSE0 **%SW63** HSBY_REVERSE1 **%SW64** HSBY_REVERSE2 **%SW65** HSBY_REVERSE3	Mot de transfert	Ces 4 mots peuvent être modifiés par l'utilisateur de la première section de la tâche maître. Ils sont ensuite transférés automatiquement du processeur redondant à l'automate primaire. Ils peuvent être lus sur l'automate primaire et utilisés comme paramètres de l'application primaire.	0

Description des mots système Quantum %SW98 à %SW100

Description détaillée

Description des mots système %SW98 à %SW100.

Mot Symbole	Fonction	Description	Etat initial

%SW98
CRA_COMPAT_LOW
	Registre d'état de compatibilité faible CRA	Signification des différents bits du mot %SW98 :
		● %SW98.0 n'est pas utilisé et est réglé sur 0 par défaut.
		● %SW98.1 à %SW98.15
		  ● =0 indique que les stations 2 à 16 ne sont pas compatibles.
		  ● =1 indique que les stations 2 à 16 sont compatibles.
			0

%SW99
CRA_COMPAT_HIGH
	Registre d'état de compatibilité élevée CRA	Signification des différents bits du mot %SW99 :
		● %SW99.0 à %SW99.15
		  ● =0 indique que les stations 17 à 32 ne sont pas compatibles.
		  ● =1 indique que les stations 17 à 32 sont compatibles.
			0

%SW100
CCOTF_COUNT
	Registre d'état de comptage CCOTF	Signification des différents bits du mot %SW100 :
		● XXYY
		  ● XX est incrémenté chaque fois qu'une configuration d'E/S est effectuée en mode RUN sur une station d'E/S distantes,
		  ● YY est incrémenté chaque fois qu'une configuration d'E/S est effectuée en mode RUN sur un rack local.
			0

Description des mots système Quantum %SW110 à %SW179

Description détaillée

Description des mots système %SW110 à %SW179 ; ces mots sont actifs sur les automates Quantum 140 CPU 6•• •••.

Mot Symbole	Fonction	Description	Etat initial
%SW110	Taille de la zone mémoire non restreinte pour %M	Ce mot système donne des informations sur la taille de la zone mémoire non restreinte pour %M.	0
%SW111	Taille de la zone mémoire non restreinte pour %MW	Ce mot système donne des informations sur la taille de la zone mémoire non restreinte pour %MW.	0
%SW128 NB_P502_CNX	Nombre de connexions ouvertes	L'octet de poids fort de ce mot indique le nombre de connexions TCP ouvertes sur la liaison Ethernet port TCP/IP 502.	0
%SW129 NB_DENIED_CNX	Nombre de connexions refusées	Ce mot indique le nombre de connexions TCP refusées sur la liaison Ethernet port TCP/IP 502.	0
%SW130 NB_P502_REF	Nombre de messages refusés	Ce mot indique le nombre de messages TCP refusées sur la liaison Ethernet port TCP/IP 502.	0
%SW132 et **%SW133** NB_SENT_MSG	Nombre de messages envoyés	Ce double mot %SDW132 indique le nombre de messages envoyés sur la liaison Ethernet port TCP/IP 502.	0
%SW134 et **%SW135** NB_RCV_MSG	Nombre de messages reçus	Ce double mot %SDW134 indique le nombre de messages reçus sur la liaison Ethernet port TCP/IP 502.	0
%SW136 NB_IOS_CNX	Nombre d'équipements scrutés	Ce mot indique le nombre d'équipements scrutés sur la liaison Ethernet port TCP/IP 502.	0
%SW137 NB_IOS_MSG	Nombre de messages IO Scanning reçus	Ce mot indique le nombre de messages du service IO Scanning reçus par seconde sur la liaison Ethernet port TCP/IP 502.	0
%SW138 GLBD_ERROR	Erreur de cohérence des Global Data	Erreur de cohérence des Global Data	0
%SW139 BW_GLBD_IOS	Charge des services Global Data et IO Scanning	L'octet de poids faible de ce mot mesure le pourcentage de charge relative au IO Scanning. L'octet de poids fort de ce mot mesure le pourcentage de charge relative au Global Data.	0

Mot **Symbole**	**Fonction**	**Description**	**Etat initial**
%SW140 BW_OTHER_MSG	Charge du service messagerie et autres services	L'octet de poids faible de ce mot mesure le pourcentage de charge relative au service messagerie. L'octet de poids fort de ce mot mesure le pourcentage de charge relative aux autres services.	0
%SW141 et %SW142 IP_ADDR	Adresse IP	Ce double mot %SDW141 reçoit l'adresse IP de la liaison Ethernet.	0
%SW143 et %SW144 IP_NETMASK	Masque de sous-réseau IP	Ce double mot %SDW143 reçoit le masque sous-réseau de la liaison Ethernet.	0
%SW145 et %SW146 IP_GATEWAY	Adresse passerelle Ethernet par défaut	Ce double mot %SDW145 reçoit l'adresse de la passerelle Ethernet par défaut.	0
%SW147 à %SW149 MAC_ADDR1 à 3	Adresses MAC	Les mots %SW147, %SW148,%SW149 codent respectivement les adresses MAC 1, MAC 2 et MAC 3.	0
%SW150	Version du coprocesseur	Ce mot code la version coprocesseur pour les automates de type 140 CPU 671-60. La version s'affiche au format héxadécimal.	0
%SW151 BOARD_STS	Etat de la liaison Ethernet	Ce mot code l'état de la liaison Ethernet. ● Bit 0 =0 si liaison Ethernet est arrêtée ● Bit 1 =0 ● Bit 2 : 0= mode Haf duplex, 1=full duplex ● Bit 3 =0 ● Bit 4 à 11 : =7 pour Quantum, =6 pour Hot Stand by Quantum ● Bit 12 : 0 = liaison 10 Mbits, 1= liaison 100 Mbits ● Bit 13 : 0 = liaison 10/100Base-TX (paire torsadée) ● Bit 14 : 0 ● Bit 15 : 0 = liaison Ethernet inactive, 1= liaison Ethernet active	0

Mot **Symbole**	**Fonction**	**Description**	**Etat initial**
%SW160 à %SW167 REFRESH_IO	Etat de fonctionnement des équipements par IO scanning	Les bits des mots %SW160 à %SW167 sont associés aux équipements scrutés par E/S. Le bit est défini sur 0 si l'équipement est en défaut, il est défini sur 1 si l'équipement fonctionne correctement. %SW160.0 : équipement N° 1. %SW160.1 : équipement N° 2. ..... %SW167.15 : équipement N° 128. **Remarque :** Ces mots système sont uniquement disponibles pour les coprocesseurs Quantum et non pour les modules NOE.	-
%SW168 à %SW171 VALID_GD	Etat de fonctionnement des Global Data	Les bits des mots %SW168 à %SW171 sont associés aux Global Data. Le bit est défini sur 0 si l'équipement est en défaut, il est sur 1 si l'équipement fonctionne correctement. %SW168.0 : équipement N° 1. %SW168.1 : équipement N° 2. ..... %SW171.15 : équipement N° 64.	-

Description des mots système Quantum %SW180 à %SW640

Description détaillée

Description des mots système %SW180 à %SW640.

Mot Symbole	Fonction	Description	Etat initial
%SW180 à %SW339 IOHEALTHij i=1..32, j=1..5	Etat de fonctionnement des modules automates	Les mots %SW180 à %SW339 sont associés aux stations automates : 5 mots par station correspondant aux racks 1 à 5 de chaque station. %SW180 : état de fonctionnement des modules du rack 1 de la station 1. %SW181 : état de fonctionnement des modules du rack 2 de la station 1. ..... %SW185 : état de fonctionnement des modules du rack 1 de la station 2. %SW186 : état de fonctionnement des modules du rack 2 de la station 2. ..... Les bits de 0 à 15 de chacun de ces mots sont associés aux modules situés aux positions 16 à 1 de ces racks. Le bit est défini sur 0 si le module est en défaut, il est sur 1 si le module fonctionne correctement. **Exemple :** %SW185.5 =0 Le module situé dans l'emplacement 11 du rack 1 de la station 2 est en défaut. **Remarque** : les modules 140 XBE 100 00 nécessitent une gestion toute particulière. Ces mots ne sont pas disponibles sur les automates de sécurité.	0
%SW340 MB+DIOSLOT	Numéro d'emplacement du processeur avec liaison Modbus Plus	Numéro d'emplacement du processeur intégrant la liaison Modbus Plus pour la connexion au premier réseau DIO. Le numéro d'emplacement est codé de 0 à 15. Ce mot n'est pas disponible sur les automates de sécurité Quantum.	-

Mot Symbole	Fonction	Description	Etat initial
%SW341 à %SW404 MB+IOHEALTHi i=1..64	Etat de fonctionnement des modules de stations distribuées du premier réseau DIO	Les mots %SW341 à %SW404 sont associés aux stations distribuées (DIO) : 64 mots associés aux 64 stations DIO du premier réseau. %SW341 : état de fonctionnement des modules de la station 1. %SW342 : état de fonctionnement des modules de la station 2. ..... %SW404 : état de fonctionnement des modules de la station 64. Les bits de 0 à 15 de chacun de ces mots sont associés aux modules situés aux positions 16 à 1 de ces stations. Le bit est défini sur 0 si le module est en défaut, il est sur 1 si le module fonctionne correctement. **Exemple :** %SW362.5 =0 Le module situé dans l'emplacement 11 de la station 22 du premier réseau DIO est en défaut. **Remarque** : pour les modules 140 CRA 2•• •••, la valeur de ce bit n'est pas significative, elle est toujours définie sur 0. Ces mots ne sont pas disponibles sur les automates de sécurité.	-
%SW405 NOM1DIOSLOT	Numéro d'emplacement du premier module interface réseau DIO	Numéro d'emplacement du module 140 NOM 2•• pour la connexion au deuxième réseau DIO. Le numéro d'emplacement est codé de 0 à 15. Ce mot n'est pas disponible sur les automates de sécurité Quantum.	-
%SW406 à %SW469 NOM1DIOHEALTHi i=1..64	Etat de fonctionnement des modules de stations distribuées du second réseau DIO	Les mots %SW406 à %SW469 sont associés aux stations distribuées (DIO) : 64 mots associés aux 64 stations DIO du second réseau. %SW406 : état de fonctionnement des modules de la station 1. %SW407 : état de fonctionnement des modules de la station 2. ..... %SW469 : état de fonctionnement des modules de la station 64. Les bits de 0 à 15 de chacun de ces mots sont associés aux modules situés aux positions 16 à 1 de ces stations. Le bit est défini sur 0 si le module est en défaut, il est sur 1 si le module fonctionne correctement. **Exemple :** %SW412.5 =0 Le module situé dans l'emplacement 11 de la station 7 du second réseau DIO est en défaut. **Remarque** : pour les modules 140 CRA 2•• •••, la valeur de ce bit n'est pas significative, elle est toujours définie sur 0. Ces mots ne sont pas disponibles sur les automates de sécurité.	-
%SW470 NOM2DIOSLOT	Numéro d'emplacement du second module interface réseau DIO	Numéro d'emplacement du module 140 NOM 2•• pour la connexion au troisième réseau DIO. Le numéro d'emplacement est codé de 0 à 15. Ce mot n'est pas disponible sur les automates de sécurité Quantum.	-

Mot **Symbole**	**Fonction**	**Description**	**Etat initial**
%SW471 à %SW534 NOM2DIOHEALTHi i=1..64	Etat de fonctionnement des modules de stations distribuées du troisième réseau DIO	Les mots %SW471 à %SW534 sont associés aux stations distribuées (DIO) : 64 mots associés aux 64 stations DIO du troisième réseau. %SW471 : état de fonctionnement des modules de la station 1. %SW472 : état de fonctionnement des modules de la station 2. ..... %SW534 : état de fonctionnement des modules de la station 64. Les bits de 0 à 15 de chacun de ces mots sont associés aux modules situés aux positions 16 à 1 de ces stations. Le bit est défini sur 0 si le module est en défaut, il est sur 1 si le module fonctionne correctement. **Exemple :** %SW520.5 =0 Le module situé dans l'emplacement 11 de la station 86 du troisième réseau DIO est en défaut. **Remarque** : pour les modules 140 CRA 2•• •••, la valeur de ce bit n'est pas significative, elle est toujours définie sur 0. Ces mots ne sont pas disponibles sur les automates de sécurité.	-

Mot Symbole	Fonction	Description	Etat initial
%SW535 RIOERRSTAT	Erreur RIO au démarrage	Ce mot stocke le code d'erreur de démarrage. Il est toujours défini sur 0 lorsque le système est en marche. En cas d'erreur, l'automate ne démarre pas, mais génère un code d'état d'arrêt. 01 : Longueur d'affectation des E/S 02 : Numéro de lien d'E/S décentralisée 03 : Nombre de stations dans l'affectation des E/S 04 : Checksum d'affectation des E/S 10 : Longueur du descripteur de station 11 : Numéro de station d'E/S 12 : Temps d'autonomie de la station 13 : Numéro de port ASCII 14 : Nombre de modules de la station 15 : Station déjà configurée 16 : Port déjà configuré 17 : Plus de 1 024 points de sortie 18 : Plus de 1 024 points d'entrée 20 : Adresse d'emplacement de module 21 : Adresse du châssis de module 22 : Nombre d'octets de sortie 23 : Nombre d'octets d'entrée 25 : Premier numéro de référence 26 : Second numéro de référence 28 : Bits internes hors limite des 16 bits 30 : Module de sortie impair dépareillé 31 : Module d'entrée impair dépareillé 32 : Référence de module impair dépareillé 33 : Référence 1x après le registre 3x 34 : Référence du module factice déjà utilisé 35 : Le module 3x n'est pas factice 36 : Le module 4x n'est pas factice	-

Mot Symbole	Fonction	Description	Etat initial

%SW536
CAERRCNT0
%SW537
CAERRCNT1%SW538
CAERRCNT2
	Etat de la communication sur le câble A	Les mots %SW536 à %SW538 sont des mots d'erreur de communication sur le câble A.
		• %SW536 :
		  • octet de poids fort : compte les erreurs de trame
		  • octet de poids faible : compte les dépassements du récepteur DMA
		• %SW537 :
		  • octet de poids fort : compte les erreurs de réception
		  • octet de poids faible : compte les réceptions de stations incorrectes
		• %SW538 :
		  • %SW538.15 = 1, trame trop courte
		  • %SW538.14 = 1, pas de fin de trame
		  • %SW538.3 = 1, erreur CRC
		  • %SW538.2 = 1, erreur d'alignement
		  • %SW538.1 = 1, erreur de dépassement
		  • %SW538.13 à 4 et 0 sont inutilisés
			-

%SW539
CBERRCNT0
%SW540
CBERRCNT1 to %SW541
CBERRCNT2
	Etat de la communication sur le câble B	Les mots %SW539 à %SW541 sont des mots d'erreur de communication sur le câble B.
		• %SW539 :
		  • octet de poids fort : compte les erreurs de trame
		  • octet de poids faible : compte les dépassements du récepteur DMA
		• %SW540 :
		  • octet de poids fort : compte les erreurs de réception
		  • octet de poids faible : compte les réceptions de stations incorrectes
		• %SW541 :
		  • %SW541.15 = 1, trame trop courte
		  • %SW541.14 = 1, pas de fin de trame
		  • %SW541.3 = 1, erreur CRC
		  • %SW541.2 = 1, erreur d'alignement
		  • %SW541.1 = 1, erreur de dépassement
		  • %SW541.13 à 4 et 0 sont inutilisés
			-

Mot Symbole	Fonction	Description	Etat initial

%SW542
GLOBERRCNT0
%SW543
GLOBERRCNT1
%SW544
GLOBERRCNT2
	Etat de communication globale	Les mots %SW542 à %SW544 sont des mots d'erreur de communication globale.
		● %SW542 : affiche l'état de la communication globale.
		  ● %SW542.15 = 1, communication en fonctionnement correct
		  ● %SW542.14 = 1, communication sur câble A en fonctionnement correct
		  ● %SW542.13 = 1, communication sur câble B en fonctionnement correct
		  ● %SW542.11 à 8= compteur des communications perdues
		  ● %SW542.7 à 0 = compteur totalisateur de nouvelle tentative
		● %SW543 : est le compteur totalisateur global des erreurs pour le câble A :
		  ● octet de poids fort : compte les erreurs détectées
		  ● octet de poids faible : compte les « non réponses »
		● %SW544 : est le compteur totalisateur global des erreurs pour le câble B :
		  ● octet de poids fort : compte les erreurs détectées
		  ● octet de poids faible : compte les « non réponses »
			-

%SW545 à %SW547
MODUNHEALTH1
IOERRCNT1
IORETRY1
	Etat de la station locale	Pour les automates où la station 1 est réservée pour les entrées/sorties locales, les mots d'état %SW545 à %SW547 sont utilisés de la façon suivante.
		● %SW545 : état de la station locale
		  ● %SW545.15 = 1, tous les modules ont un fonctionnement correct
		  ● %SW545.14 à 8 = non utilisés, toujours à 0
		  ● %SW545.7 à 0 = nombre de fois où le module a été vu comme défectueux, le compteur reboucle à 255
		● %SW546 : est utilisé comme compteur des erreurs de bus d'entrées/sorties 16 bits
		● %SW547 : est utilisé comme compteur de répétition de bus d'entrées/sorties 16 bits
			-

Mot **Symbole**	**Fonction**	**Description**	**Etat initial**

%SW548 à %SW640
MODUNHEALTHi
IOERRCNTi
IORETRYi
(i=2..32)
	Etat des stations décentralisées	Les mots %SW548 à %SW640 permettent de décrire l'état des stations décentralisées. Trois mots d'état sont utilisés pour chaque station.
		● %SW548 : affiche l'état de la communication globale de la station 2 :
		  ● %SW548.15 = 1, communication en fonctionnement correct
		  ● %SW548.14 = 1, communication sur câble A en fonctionnement correct
		  ● %SW548.13 = 1, communication sur câble B en fonctionnement correct
		  ● %SW548.11 à 8= compteur des communications perdues
		  ● %SW548.7 à 0 = compteur totalisateur de nouvelle tentative
		● %SW549 : est le compteur totalisateur global des erreurs pour le câble A station 2 :
		  ● octet de poids fort : compte les erreurs détectées
		  ● octet de poids faible : compte les « non réponses »
		● %SW550 : est le compteur totalisateur global des erreurs pour le câble B station 2 :
		  ● octet de poids fort : compte les erreurs détectées
		  ● octet de poids faible : compte les « non réponses »
		Les mots :
		%SW551 à 553 sont affectés à la station 3
		%SW554 à 556 sont affectés à la station 4
		.....
		%SW638 à 640 sont affectés à la station 32
			-

6.5 Mots système spécifiques au Modicon M340

Description des mots système %SW142 à %SW145, %SW146 et %SW147, %SW150 à %SW154, %SW160 à %SW167

Description détaillée

Description des mots système %SW142 à %SW145, %SW146 et %SW147, %SW150 à %SW154, %SW160 à %SW167 :

Mot Symbole	Fonction	Description	Etat initial
%SW142 à %SW145	Modicon M340	Inhibe l'erreur générée par le système lorsqu'un équipement configuré sur le bus CANopen n'est pas présent. Cette inhibition peut être gérée avec les 4 mots système %SW142,143,144,145. Ces mots système mettent en œuvre une liste de bits indiquant l'erreur de nœud CANopen à inhiber : ● Le bit 0 de %SW142 concerne l'équipement à l'adresse de nœud 1. ● Le bit 1 de %SW142 concerne l'équipement à l'adresse de nœud 2. ● ... ● Le bit 15 de %SW145 concerne l'équipement à l'adresse de nœud 64. Valeurs de bit : ● Si le bit est réglé sur 0 et que l'équipement ne soit pas présent, une erreur est générée. ● Si le bit est réglé sur 1 et que l'équipement ne soit pas présent, aucune erreur n'est générée. **NOTE :** la valeur par défaut est 0. **NOTE :** cette inhibition peut être effectuée en temps réel, mais pour qu'elle soit prise en compte, le maître CANopen doit être réinitialisé (en réglant le bit 5 du mot de sortie .%QW0.0.2.0 sur 1).	-
%SW146 et %SW147	Modicon M340	Ces 2 mots système contiennent le numéro de série unique de la carte SD (32 bits). S'il n'y a pas de carte SD ou que la carte n'est pas reconnue, les deux mots système sont réglés sur 0. Cette information peut être utilisée pour protéger une application contre la copie.	-

Mot **Symbole**	**Fonction**	**Description**	**Etat initial**
%SW150 à %SW154	CANopen Modicon M340	Informations concernant le dernier transfert d'abandon du SDO : • %SW150 : mot de poids faible du code d'arrêt SDO. • %SW151 : mot de poids fort du code d'arrêt SDO. • %SW152 : numéro d'abonné du transfert SDO. • %SW153 : numéro d'index du transfert SDO. • %SW154 : numéro de sous-index du transfert SDO.	-
%SW160 à %SW167 PREMRACK0 à PREMRACK7	Erreur des racks 0 à 7 pour Premium et Modicon M340	Les mots %SW160 à %SW167 sont associés aux racks 0 à 7 respectivement. Les bits de 0 à 15 de chacun de ces mots sont associés aux modules situés aux positions 0 à 15 de ces racks. Le bit est à 0 si le module est défaillant et à 1 si le module fonctionne correctement. **Exemple :** %SW163.5=0 Le module situé à la position 5 du rack 3 est défaillant. Dans les cas des demi-racks, 2 demi-racks contigus forment un rack complet normal, référencé par un seul Swi.	-

Description des données

III

Dans cette partie

Cette partie décrit les différents types de données qu'il est possible d'utiliser dans un projet, et décrit comment les mettre en oeuvre.

Contenu de cette partie

Cette partie contient les chapitres suivants :

Présentation générale des données

7

Objet de ce chapitre

Ce chapitre présente de façon très générale:

- les différents types de données,
- les instances de données,
- les références de données.

Contenu de ce chapitre

Ce chapitre contient les sujets suivants :

Généralités

Présentation

Une **donnée** désigne un d'objet qui peut être instancié tel que :

- une variable,
- un bloc fonction.

Les données sont définies en trois phases qui sont :

- la phase **types de données**, dans laquelle est précisée :
  - sa catégorie,
  - son format.
- la phase **instances de données**, dans laquelle est défini son emplacement mémoire et sa propriété qui est:
  - localisée ou,
  - non localisée.
- la phase **références de données**, dans laquelle est défini son moyen d'accès :
  - par valeur immédiate,
  - par nom,
  - par adresse.

Illustration

Voici les trois phases caractérisant les données :

Instancier une donnée consiste à partir de son type à lui allouer un emplacement mémoire.

Référencer une donnée consiste à lui définir une référence (nom, adresse, etc...) permettant de l'atteindre en mémoire.

Présentation des familles de types de données

Présentation

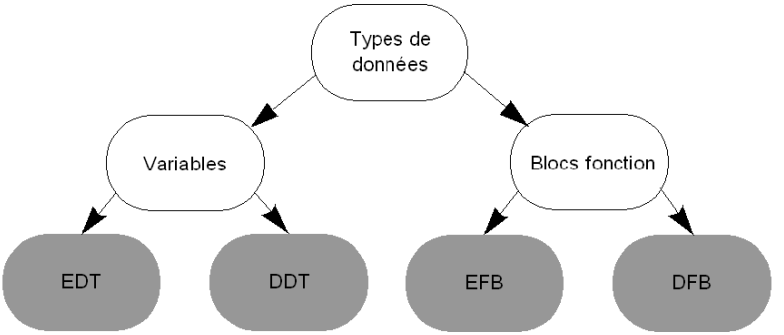
Un **type de donnée** est une information logicielle qui spécifie pour une donnée:

- sa structure,
- son format,
- la liste de ses attributs,
- son comportement.

Ces propriétés sont partagées par toutes les instances du type de donnée.

Illustation

Les familles de types de données sont classées dans différentes catégories (gris foncé).



Définitions

Familles de types de données et leurs définitions.

Famille	Définition
Structures	Types de données élémentaires (Elementary data types) tels que: ● Bool ● Int ● Byte ● Mot ● Dword ● etc.

Tableaux	Types de données dérivés (Derived data types) tels que:
	● tableaux, qui contiennent des éléments de même type:
	  ● tableaux de Bool (tableaux d'EDT),
	  ● tableaux de tableaux (tableaux de DDT),
	  ● tableaux de structures (tableau de DDT),
	● structures, qui contiennent des éléments de type différents :
	  ● structures de Bool, Word, etc. (structures EDT)
	  ● structures de tableaux, structures de structures, structures de tableaux/structures (structures de DDT),
	  ● structures de Bool, structures de tableaux, etc. (structures EDT et DDT)
	  ● structures concernant les données d'entrées/de sorties (structures d'IODDT),
	  ● Structures contenant des variables restituant les propriétés et l'état d'une action ou transition d'un diagramme fonctionnel en séquence (Sequential Function Chart).

EFB	Blocs fonction élémentaires écrits en langage C. Ils comprennent : ● des variables d'entrées, ● des variables internes, ● des variables de sorties, ● un algorithme de traitement.
DFB	Blocs fonctions utilisateurs (Derived function blocs) écrits en langage d'automatisme (Litteral Structuré, Liste d'instructions, etc...). Ils comprennent : ● des variables d'entrées, ● des variables internes, ● des variables de sorties, ● un algorithme de traitement.

Présentation des instances de données

Présentation

Une **instance de données** est une entité fonctionnelle individuelle, qui possède toutes les caractéristiques du type de données duquel elle dépend.

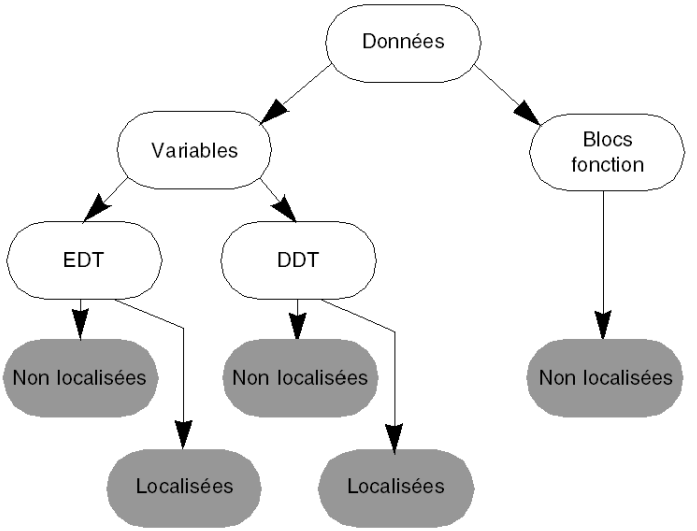
Une ou plusieurs instances peuvent être rattachées à un type de données.

L'instance de données peut avoir une allocation mémoire:

- non localisée ou
- localisée

Illustration

Allocation mémoire des instances (gris foncé) appartenant aux différents types.



Définitions

Définition des allocations mémoire des instances de données.

Instance de données	Définition
Non localisées	L'emplacement mémoire de l'instance est alloué automatiquement par le système, il peut changer à chaque génération de l'application. L'instance est repérée par un nom (symbole) choisi par l'utilisateur.
Localisées	L'emplacement mémoire de l'instance est fixe, il est prédéfini et ne change jamais. L'instance est repérée par un nom (symbole) choisi par l'utilisateur et une adresse topologique définie par le constructeur, ou uniquement par l'adresse topologique constructeur.

Présentation des references de données

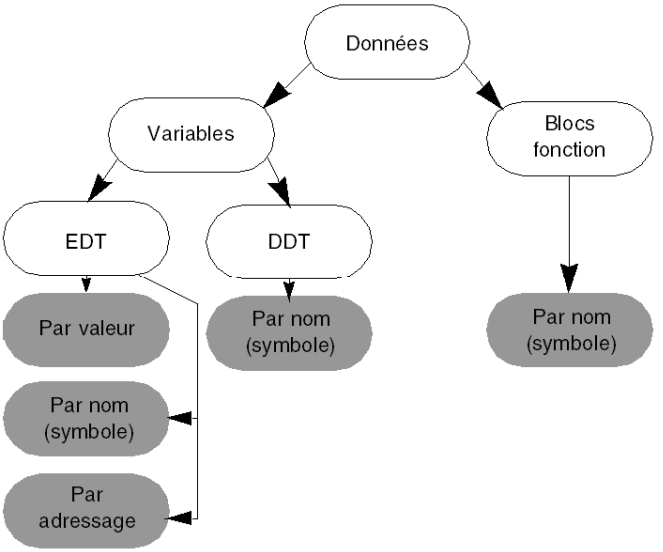
Présentation

Une **référence de données** permet à l'utilisateur d'accéder à l'instance de cette donnée soit par:

- valeur immédiate, vrai uniquement pour les données de type EDT
- adressage, vrai uniquement pour les données de type EDT
- nom (symbole), vrai pour tous les types de données EDT, DDT, EFB, DFB ainsi que les objets SFC.

Illustration

Références de données possibles suivant le type de données (gris foncé).



Règles de syntaxe pour les noms Type\Instance

Présentation

La syntaxe des noms de types et de variables peut s'effectuer avec ou sans l'utilisation du jeu de caractères étendu. Cette option peut être sélectionnée sous l'onglet **Extensions de langage** du menu **Outils ->Options du projet**.

- Si l'option **Jeu étendu de caractères autorisé** est sélectionnée, l'application est conforme à la norme CEI.
- Si l'option **Jeu étendu de caractères autorisé** n'est pas sélectionnée, l'utilisateur bénéficie de quelques souplesses, mais l'application n'est pas conforme à la norme CEI.

Le jeu étendu de caractères utilisé pour les noms saisis dans l'application concernent :

- les types blocs fonctions utilisateur DFB ou les types de données dérivés DDT,
- les éléments internes composant un type de données bloc fonction DFB/EFB ou un type de données dérivé DDT,
- les instances de données,

Si la case à cocher "Jeu étendu..." est sélectionnée

Les noms saisis sont des chaines composées de caractères alphanumériques, du caractère Underscore.

Les régles sont les suivantes:

- le premier caractère du nom est un caractère alphabétique ou le caractère de soulignement,
- deux caractères de soulignement ne peuvent être consécutifs.

Si la case à cocher "Jeu étendu..." n'est pas sélectionnée

Les noms saisis sont des chaines composées de caractères alphanumériques, du caractère Underscore.

Des caractères supplémentaires sont autorisés tels que:

- les caractères correspondant aux codes ASCII 192 à 223 (à l'exception du code 215),
- les caractères correspondant aux codes ASCII 224 à 255 (à l'exception du code 247).

Les régles sont les suivantes:

- le premier caractère du nom est un caractère **alphanumérique** ou le caractère de soulignement,
- les caractères de soulignement **peuvent être** consécutifs.

Types de données

8

Objet de ce chapitre

Ce chapitre décrit tous les types de données qu'il est possible d'utiliser dans une application.

Contenu de ce chapitre

Ce chapitre contient les sous-chapitres suivants :

8.1 Types de données élémentaires (EDT) au format Binaire

Objet de cette section

Cette section décrit le type de données au format binaire. qui sont :

- les types Booléens,
- les types Entiers,
- le type Heure.

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Présentation des types de données au format binaire

Présentation

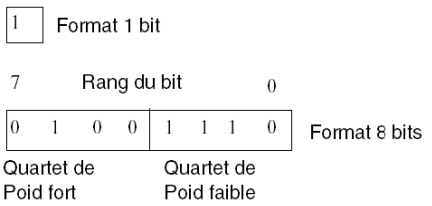
Les types de données au format Binaire appartiennent à la famille EDT (Elementary Data Type [Type de données élémentaires]), qui comprend des types de données **simples** plutôt que dérivées (tableaux, structures, blocs fonctions).

Rappel sur le format binaire

Une donnée au format binaire est constituée d'un ou de plusieurs bits ou chacun d'entre eux est représenté par un des chiffres de la base 2 soit **0** ou **1**.

L'échelle de la donnée dépend du nombre de bit(s) qui la compose.

Exemple :



Une donnée peut être:

- signée. Dans ce cas le bit de rang le plus haut est le bit de signe :
  - 0 indique une valeur positive,
  - 1 indique une valeur négative.

  La plage des valeurs est:

$$[-2^{\langle Bits-1\rangle}, 2^{\langle Bits-1\rangle}-1]$$

- non signée. Dans ce cas tous les bits représentent la valeur
  La plage des valeurs est:

$$[0, 2^{Bits}-1]$$

Bits=nombre de bits (format).

Types de données au format binaire

Liste des types de données:

Type	Désignation	Format (bits)	Valeur par défaut
BOOL	Booléen	8	0=(False)
EBOOL	Booléen avec détection de fronts et forçage	8	0=(False)
INT	Entier	16	0
DINT	Entier double	32	0
UINT	Entier non signé	16	0
UDINT	Entier double non signé	32	0
TIME	Entier double non signé	32	T=0s

Types booléens

Présentation

Il existe deux types Booléens. qui sont:

- le type BOOL, qui contient uniquement la valeur FALSE (=0) ou TRUE (=1),
- le type EBOOL, qui contient la valeur FALSE (=0) ou TRUE (=1) mais aussi des informations concernant la gestion des fronts (montants ou descendants) et le forçage.

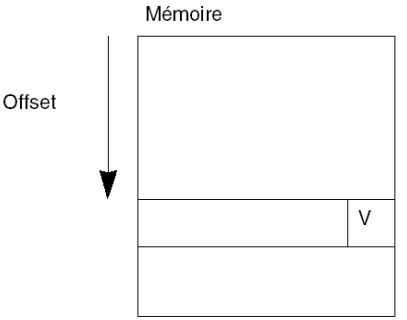
Principe du type BOOL

Ce type occupe un octet en mémoire, mais la valeur est stockée seulement dans un bit.

La valeur pas défaut de ce type est FALSE (=0).

Il est accessible par une adresse contenant l'offset sur l'octet correspondant:

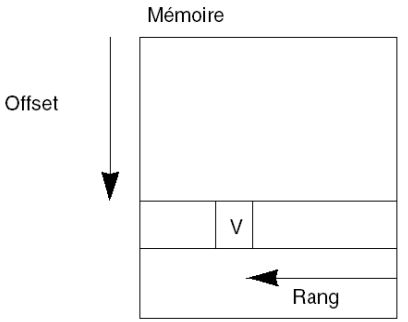
Adressage:



Dans le cas **du bit extrait de mot**, il est accessible par une adresse contenant les informations suivantes:

- un offset sur l'octet correspondant,
- le rang définissant sa position dans le mot

Adressage:



Principe du type EBOOL

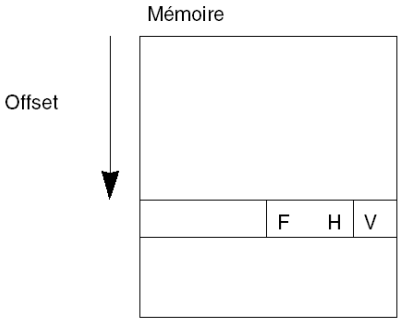
Ce type occupe un octet en mémoire dans lequel il y a :

- le bit pour de la valeur (V),
- le bit historique (H) pour la gestion des fronts (montants ou descendants). ,
- le bit contenant l'état de forçage (F). Egal à 0 si l'objet n'est pas forcé et égal à 1 si l'objet est forcé.

La valeur par défaut des bits associés au type EBOOL est FALSE (=0).

Il est accessible par une adresse spécifiant l'offset sur l'octet correspondant.

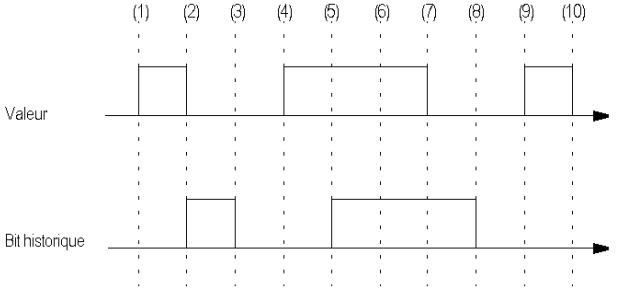
Adressage:



Chronogrammeet historique

Le chronogramme ci-dessous présente le principe des états des bits (valeur et historique) associés au type EBOLL.

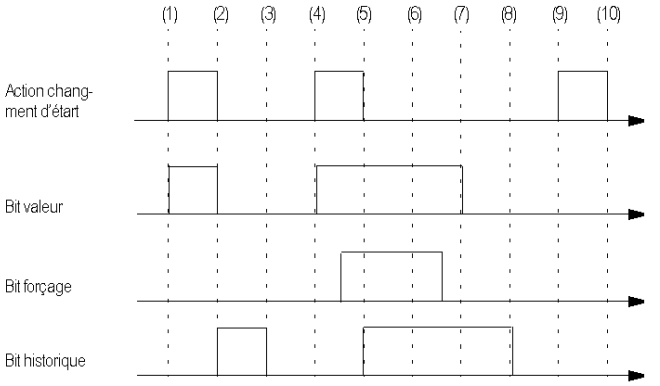
Les fronts montants du bit valeur (1, 4) sont copiées dans le bit historique au cycle automate suivant (2, 5). Les fronts descendants du bit valeur (2, 7) sont copiées dans le bit historique au cycle automate suivant (3, 8).



Chronogramme et forcage

Le chronogramme ci-dessous présente le principe des états des bits (valeur, historique, forçage) associés au type EBOLL.

Les fronts montants du bit valeur (1, 4) sont copiées dans le bit historique au cycle automate suivant (2, 5). Les fronts descendants du bit valeur (2, 7) sont copiés dans le bit historique au cycle automate suivant (3,8). Entre (4 et 5) le bit de forçage est égal à 1, les bits valeur et historique sont maintenus à 1.



Variables automate appartenant aux types booléens

Liste des variables

Variable	Type
Bit interne	EBOOL
Bit système	BOOL
Bit extrait de mot	BOOL
Entrées %I	
Bit erreur module	BOOL
Bit erreur voie	BOOL
Bit d'entrée	EBOOL
Sorties %Q	
Bit de sortie	EBOOL

Compatibilité entre BOOL et EBOOL

Les opérations autorisées entre ces deux types de variables sont:

- la copie de valeur,
- la copie d'adresse.

Copie entre types

	Destination BOOL	Destination EBOOL
Source BOOL	Oui	Oui
Source EBOOL	Oui	Oui

Compatibilité entre les paramètres des fonctions élémentaires (EF)

Paramètre effectif (externe à l'EF)	Paramètre formel BOOL (interne à l'EF)	Paramètre formel EBOOL (interne l'EF)
BOOL	Oui	Non
EBOOL	In ->Oui In-Out ->Non Out -> Oui	Oui

Compatibilité entre les paramètres des blocs fonctions (EFB\DFB)

Paramètre effectif (externe au FB)	**Paramètre formel BOOL (interne au FB)**	**Paramètre formel EBOOL (interne au FB)**
BOOL	Oui	In ->Oui In-Out ->Non Out -> Oui
EBOOL	In ->Oui In-Out ->Non Out -> Oui	Oui

Compatibilité entre variables de tableau

	Destination ARRAY[i..j) OF BOOL	**Destination ARRAY[i..j) OF EBOOL**
Source ARRAY[i..j) OF BOOL	Oui	Non
Source ARRAY[i..j) OF EBOOL	Non	Oui

Compatibilité entre variables statique

	Adressage direct BOOL (%MW:xi)	**Adressage direct EBOOL (%Mi)**
Variable declarée BOOL (Var:BOOL)	Oui	Non
Variable déclarée EBOOL (Var:EBOOL)	Non	Oui

Compatibilités

Le type de données EBOOL suit les règles suivantes :

- Une variable de type EBOOL ne peut être passée comme paramètre d'entrée/sortie de type BOOL.
- Les tableaux de EBOOL ne peuvent être passées comme paramètres de type ANY d'un FFB.
- Les tableaux de BOOL et de EBOOL ne sont pas compatibles pour l'instruction d'affectation (même règle que pour les paramètres de FFB).
- Sur Quantum :
  - Les variables localisées de type EBOOL ne peuvent être passées comme paramètres d'entrées/sorties de type EBOOL.
  - Les tableaux de EBOOL ne peuvent être passés comme paramètres d'un DFB.

Types Entier

Présentation

, qui sont :

- la base 10 (décimal) par défaut. Dans ce cas, la valeur est signée ou non signée. Cela dépend du type de l'entier.
- la base 2 (binaire). Dans ce cas, la valeur est non signée et le préfixe est **2#**.
- la base 8 (octal). Dans ce cas la valeur est non signée, le préfixe est **8#**.
- la base 16 (hexadécimal). Dans ce cas la valeur est non signée, le préfixe est **16#**.

NOTE : En représentation décimale si le type choisi est signé, la valeur peut être précédée du signe + ou du signe - (le signe + est optionnel).

Type Entier (INT)

Type signé ayant un format sur 16 bits.

Ce tableau donne la plage dans chaque base.

Base	**de...**	**à...**
Decimal	-32768	32767
Binaire	2#1000000000000000	2#0111111111111111
Octale	8#100000	8#077777
Hexadécimal	16#8000	16#7FFF

Type Entier double (DINT)

Type signé ayant un format sur 32 bits.

Ce tableau donne la plage dans chaque base.

Base	**de...**	**à...**
Decimal	-2147483648	2147483647
Binaire	2#10000000000000000000000000000000	2#01111111111111111111111111111111
Octale	8#20000000000	8#17777777777
Hexadécimale	16#80000000	16#7FFFFFFF

Type Entier non signé (UINT)

Type non signé ayant un format sur 16 bits.

Ce tableau donne la plage dans chaque base.

Base	de...	à...
Decimal	0	65535
Binaire	2#0	2#1111111111111111
Octal	8#0	8#177777
Hexadécimal	16#0	16#FFFF

Type Entier double non signé (UDINT)

Type non signé ayant un format sur 32 bits.

Ce tableau donne la plage dans chaque base.

Base	de...	à...
Decimal	0	4294967295
Binaire	2#0	2#11111111111111111111111111111111
Octal	8#0	8#37777777777
Hexadécimal	16#0	16#FFFFFFFF

Le type Heure

Présentation

Le type Heure **T#** ou **TIME#** est représenté par un type d'entier double non signé (UDINT) *(voir page 248)*.

Il exprime une durée en millisecondes, qui représente environ la durée maximale de 49 jours.

Les unités de temps autorisées pour la représentation de la valeur sont les suivantes :

- jours **(J)**
- heures **(H)**
- minutes **(M)**
- secondes **(S)**
- millisecondes **(MS)**

Saisie d'une valeur

Ce tableau montre les possibilités de saisie de la valeur maximale du type **Temps** en fonction des unités de temps autorisées.

Schéma	Commentaire
T#4294967295**MS**	valeur en millisecondes
T#4294967**S**_295**MS**	valeur en secondes\millisecondes
T#71582**M**_47**S**_295**MS**	valeur en minutes\secondes\millisecondes
T#1193**H**_2**M**_47**S**_295**MS**	valeur en heures\minutes\secondes\millisecondes
T#49**J**_17**H**_2**M**_47**S**_295**MS**	valeur en jours\heures\minutes\secondes\millisecondes

8.2 Types de données élémentaires (EDT) au format BCD

Objectif de la section

Cette section décrit les types de données au format BCD (Binary Coded Decimal) qui sont :

- le type Date,
- le type Time of Day (TOD),
- le type Date and Time (DT).

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Présentation des types de données au format BCD

Présentation

Les types de données au format BCD appartiennent à la famille de données élémentaires EDT (Elementary data type) qui regroupe des types de données **dits simples** et non pas composées (tableaux, structures, blocs fonction).

Rappel sur le format BCD

Le format Décimal codé Binaire (Binary coded Decimal) permet de représenter les chiffres décimaux **compris entre 0 et 9** à l'aide d'un ensemble de quatre bits (quarté).

Dans ce format, les quatre bits utilisés pour coder les nombres décimaux ont une plage de combinaisons inutilisées.

Table de correspondance:

Décimal	Binaire
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
	1010 (inutilisée)
	1011 (inutilisée)
	1100 (inutilisée)
	1101 (inutilisée)
	1110 (inutilisée)
	1111 (inutilisée)

Exemple de codage sur un format de 16 bits:

Valeur décimale 2450	2	4	5	0
Valeur Binaire	0010	0100	0101	0000

Exemple de codage sur un format de 32 bits:

Valeur décimale 78993016	7	8	9	9	3	0	1	6
Valeur Binaire	0111	1000	1001	1001	0011	0000	0001	0110

Types de données au format BCD

Trois types de données:

Type	Désignation	Echelle (bits)	Valeur par défaut
DATE	Date	32	D#1990-01-01
TIME_OF_DAY	Heure du jour	32	TOD#00:00:00
DATE_AND_TIME	Date et heure	64	DT#1990-01-01-00:00:00

Le type Date

Présentation

Le type **Date** codé sur un format de 32 bits contient les informations suivantes:

- l'année codée dans un champ de 16 bits (4 quartés de poids forts),
- le mois codé dans un champ de 8 bits (2 quartés),
- le jour codé dans un champ de 8 bits (2 quartés de poids faibles).

Représentation au format BCD de la date 2001-09-20:

Année (2001)	**Mois (09)**	**Jour (20)**
0010 0000 0000 0001	0000 1001	0010 0000

Régles de syntaxe

La saisie du type **Date** est la suivante : **D#**<Année>**-**<Mois>**-**<Jour>

Ce tableau donne les bornes inférieures\supérieures de chaque champ.

Champ	**Bornes**	**Commentaire**
Année	[1990,2099]	
Mois	[01,12]	Le 0 de gauche est toujours affiché ; il peut être omis lors de la saisie.
Jour	[01,31]	Pour les mois 01/03/05/07/08/10/12
	[01,30]	Pour les mois 04/06/09/11
	[01,29]	Pour le mois 02 (années bissextiles)
	[01,28]	Pour le mois 02 (années non bissextiles)

Exemple :

Saisie	**Commentaires**
D#2001-1-1	Le 0 de gauche du mois et jour peut être omis
d#1990-02-02	Le préfixe peut être en minuscule

Le type Time of Day (TOD)

Présentation

Le type **Time of Day** codé sur un format de 32 bits contient les informations suivantes:

- l'heure codée dans un champ de 8 bits (2 quartés de poids forts),
- les minutes codées dans un champ de 8 bits (2 quartés),
- les secondes codées dans un champs de 8 bits (2 quartés).

NOTE : Les 8 bits de poids faible sont inutilisés.

Représentation au format BCD de l'heure du jour 13:25:47:

Heure (13)	Minutes (25)	Secondes (47)	Octet de poid faible
0001 0011	0010 0101	0100 0111	Inutilisés

Régles de syntaxe

Le type Time Of Day doit être saisi comme suit :
TOD#<Heure>**:**<Minutes>**:**<Secondes>

Ce tableau donne les bornes inférieures\supérieures de chaque champ.

Champ	Bornes	Commentaire
Heure	[00,23]	Le 0 de gauche est toujours affiché ; il peut être omis lors de la saisie.
Minute	[00,59]	Le 0 de gauche est toujours affiché ; il peut être omis lors de la saisie.
Seconde	[00,59]	Le 0 de gauche est toujours affiché ; il peut être omis lors de la saisie.

Exemple :

Saisie	Commentaire
TOD#1:59:0	Les 0 de gauche des heures et secondes peuvent être omis
tod#23:10:59	Le préfixe peut être en minuscule
Tod#0:0:0	Le préfixe peut être mixte (minuscule\majuscule)

Le type Date and Time (DT)

Présentation

Le type **Date and Time** codé sur un format de 64 bits contient les informations suivantes:

- l'année codée dans un champ de 16 bits (4 quartés de poids forts),
- le mois codé dans un champ de 8 bits (2 quartés),
- le jour codé dans un champ de 8 bits (2 quartés),
- l'heure codée dans un champ de 8 bits (2 quartés),
- les minutes codées dans un champ de 8 bits (2 quartés),
- les secondes codées dans un champs de 8 bits (2 quartés).

NOTE : Les 8 bits de poids faible sont inutilisés.

Exemple : Représentation au format BCD de la date et heure 2000-09-20:13:25:47.

Année (2000)	Mois (09)	Jour (20)	Heure (13)	Minute(25)	Secondes (47)	Octet de poid faible
0010 0000 0000 0000	0000 1001	0010 0000	0001 0011	0010 0101	0100 0111	Inutilisés

Régles de syntaxe

La saisie du type **Date and Time** est la suivante:

DT#<Année>**-**<Mois>**-**<Jour>**-**<Heure>**:**<Minutes>**:**<Secondes>

Ce tableau donne les bornes inférieures\supérieures de chaque champ.

Champ	Bornes	Commentaire
Année	[1990,2099]	
Mois	[01,12]	Le 0 à gauche est toujours affiché, il peut être omis lors de la saisie
Jour	[01,31]	Pour les mois 01/03/05/07/08/10/12
	[01,30]	Pour les mois 04/06/09/11
	[01,29]	Pour le mois 02 (années bissextiles)
	[01,28]	Pour le mois 02 (années non bissextiles)
Heure	[00,23]	Le 0 à gauche est toujours affiché, il peut être omis lors de la saisie
Minute	[00,59]	Le 0 à gauche est toujours affiché, il peut être omis lors de la saisie
Seconde	[00,59]	Le 0 à gauche est toujours affiché, il peut être omis lors de la saisie

Exemple :

Saisie	Commentaire
DT#2000-1-10-0:40:0	Les 0 de gauche des mois\heure\seconde peuvent être omis
dt#1999-12-31-23:59:59	Le préfixe peut être en minuscule
Dt#1990-10-2-12:02:30	Le préfixe peut être mixte (minuscule\majuscule)

8.3 Types de données élémentaires (EDT) au format Réel

Présentation du type de données Real

Introduction

Les types de données au format Binaire appartiennent à la famille EDT (Elementary Data Type [Type de données élémentaires]), qui comprend des types de données **simples** plutôt que dérivés (tables, structures, blocs fonction).

Rappel concernant le format Real

Le format Real (virgule flottante conformément au standard ANSI/IEEE 754) est codé sur 32 bits, qui correspond aux nombres à virgule flottante à décimale unique.

Les 32 bits représentant la valeur à virgule flottante sont organisés en trois champs distincts, qui sont :

- **S**, le bit du signe qui peut prendre la valeur :
  - 0 pour un nombre à virgule flottante positif,
  - 1 pour un nombre à virgule flottante négatif.
- **e**, l'exponentiel codé dans un champ de 8 bits (nombre entier au format binaire),
- **f**, la partie à virgule fixe, codée dans un champ de 23 bits (nombre entier au format binaire).

Représentation :

La valeur de la partie à virgule fixe (Mantissa) se situe dans l'intervalle [0, 1[, et est calculée à l'aide de la formule suivante.

$F = 2^{-23} * M$

Types de nombres pouvant être représentés

Ces nombres sont les suivants :
- normalisés,
- dénormalisés,
- de valeurs infinies,
- avec des valeurs +0 et -0.

Ce tableau fournit les valeurs contenues dans les différents champs selon le type de nombre.

e	f	S	Type de nombre
]0, 255[	[0, 1[	0 ou 1	normalisé
0	[0, 1[	environ ($1,4^{E-45}$)	dénormalisé DEN
255	0	0	+ infini (INF)
255	0	1	- infini (-INF)
255	]0,1[et bit 22 = 0	0 ou 1	SNAN
255	]0,1[et bit 22 = 1	0 ou 1	QNAN
0	0	0	+0
0	0	1	-0

NOTE :

La norme CEI 559 définit deux classes de NAN (pas un nombre) : QNAN et SNAN.
- QNAN : est un NAN dont le bit 22 est défini sur 1
- SNAN : est un NAN dont le bit 22 est défini sur 0

Ils se comportent comme suit :
- Les QNAN ne déclenchent pas d'erreurs lorsqu'ils sont utilisés comme opérandes d'une fonction ou d'une expression.
- Les SNAN déclenchent une erreur lorsqu'ils sont utilisés comme opérandes d'une fonction ou d'une expression arithmétique (voir %SW17 *(voir page 175)* et %S18 *(voir page 154)*).

Ce tableau fournit la formule de calcul de la valeur du nombre à virgule flottante :

Nombre à virgule flottante	Valeur
Normalisée	$(-1)^S \times 2^{(e-127)} \times (1+f)$
Dénormalisée (DEN)	$(-1)^S \times 2^{-126} \times f$

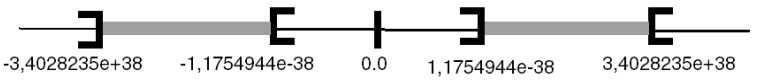
NOTE : Un nombre réel, entre -1,1754944e-38 et 1,1754944e-38, est un DEN dénormalisé. Lorsqu'un opérande est un DEN, le résultat n'est pas garanti. Les bits %SW17 *(voir page 175)* et %S18 *(voir page 154)* sont augmentés, sauf pour le Modicon M340. Les automates Modicon M340 peuvent utiliser les opérandes dénormalisés mais avec une perte de précision due à leur format. Le dépassement par valeur inférieure est signalé selon le fonctionnement uniquement lorsque le résultat est 0 (dépassement total par valeur inférieure) ou lorsque le résultat est dénormalisé (dépassement progressif par valeur inférieure avec perte de précision).

Type Real

Présentation :

Type	Echelle (bits)	Valeur par défaut
REAL	32	0.0

Plage de valeurs (parties grisées) :



Lorsqu'un résultat est :
- compris entre -1,1754944e-38 et 1,1754944e-38, le nombre est un DEN ;
- inférieur à -3,4028234e+38, le symbole -INF (pour -infini) s'affiche ;
- supérieur à +3,4028234e+38, le symbole INF (pour + infini) s'affiche ;
- indéfini (racine carrée d'un nombre négatif), le symbole NAN s'affiche.

Exemples d'imprécision de la valeur normalisée

7,986 est codé par l'application comme suit :

S	E=129	M=8359248
0	1000001	11111111000110101010000

A l'aide de la formule :

$$(-1)^0 \times 2^{(129-127)} \times \left(1 + \frac{8359248}{2^{23}}\right) = (7,9860000610351456 25)$$

Le nombre 7,986 doit avoir une valeur significative de :

$$\left(\frac{7,986}{2^2} - 1\right) \times 2^{23} = (8359247,872)$$

La valeur significative étant exprimée sous la forme d'un entier, elle ne peut être codée que sous la forme 8359248 (arrondie à la limite la plus proche).

Aucun nombre ne peut être codé entre les signifiants 8359247 et 8359248, ou entre les nombres réels 7,985999584197998046875 et 7,98600006103515625.

L'importance du bit de poids faible (écart) est, en précision absolue :

$$\frac{2^{(129-127)}}{2^{23}} = 2^{-21} = 0,000000476837158203125$$

L'écart devient très important pour les valeurs élevées, comme indiqué ci-dessous :

Valeur	$Range \le 2^{n-1} \le Value \le 2^n$	M=8359248
100 000 000	Entre 2^{26} et 2^{27}	$\frac{2^{26}}{2^{23}} = 2^3 = 8$
2^{127}	2^{127}	$\frac{2^{127}}{2^{23}} = 2^{104} = 2,02 \times 10^{31}$

NOTE : L'écart correspond à l'importance du bit de poids faible.

Pour obtenir la résolution souhaitée, il est nécessaire de définir la plage maximale de calcul, à l'aide de la formule suivante :

$$e = \frac{|Ln(p \times 2^{23})|}{Ln(2)}$$

p étant la précision et e l'exposant (e = E-127)

Par exemple, si la précision requise est de 0,001, la partie à virgule fixe est :

$$F = (-1)^{S} \times 2^{(e)} \times \left(1 + \frac{M}{2^{23}}\right) = 2^{14} = 16384$$

avec :

$$13 = \frac{\left|Ln(0,001 \times 2^{23})\right|}{Ln(2)}$$

Au-delà de cette limite F, la précision est perdue.

Cas typique : compteurs

La virgule flottante doit être utilisée avec précaution, notamment lorsqu'il s'agit d'ajouter un petit nombre à lui-même.

Si les incréments sont petits, le compteur ne fonctionnera pas correctement. Il donne des résultats erronés et interrompt l'incrémentation lorsque la valeur à ajouter est inférieure à celle du bit de poids faible du compteur.

Pour obtenir les valeurs correctes, il est recommander d'utiliser un double entier (`UDINT`) et de multiplier le résultat par l'incrément.

Exemple :

- Incrémentez une valeur de 0,001 entre 33000 et 1000000.
- Comptez de 33 000 000 à 1 000 000 000 (valeur x 1000) avec 1 comme incrément.
- Obtenez le résultat en multipliant la valeur par 0,001.

La précision **F** minimale par plage est :

De...à...	F (minimum)
3300...65536	0.004
65536...131072	0.008
...	...
524288...1000000	0.063

Ce compteur peut fonctionner jusqu'à 4 294 967 295 x 0,001 = 4 294 967,5 avec une précision minimale de 0,5.

NOTE : La valeur réelle ici est la valeur binaire encodée. Elle peut différer de celle affichée dans un écran d'exploitation, si elle est arrondie (4,294968e+006).

8.4 Types de données élémentaires (EDT) au format chaine de caractères

Présentation des types de données au format chaîne de caractères

Introduction

Le type de données au format chaîne de caractères appartient à la famille de données élémentaires EDT (Elementary data type) qui regroupe des types de données **dits simples** et non pas composées (tableaux, structures, blocs fonction).

Le type chaîne de caractères

Le format chaîne de caractères permet de représenter une chaîne de caractères ASCII, chaque caractère étant codé sur un format de 8 bits.

Les caractéristiques du type chaîne de caractères sont les suivantes :

- 16 caractères par défaut dans la chaîne (caractère fin de chaîne exclu),
- une chaîne est composée de caractères ASCII compris entre 16#20 et 16#FF (représentation hexadécimale).
- dans une chaîne vide, le caractère de fin de chaîne (code ASCII « NUL ») est le premier de la chaîne.
- la taille maximale d'une chaîne est de 65 535 caractères.

La taille de la chaîne de caractères peut être optimisée lors de la définition du type par la commande **STRING[<taille>]**, <taille> étant un entier non signé UINT pouvant définir une chaîne de 1 à 65 535 caractères ASCII.

NOTE : les caractères ASCII 0 à 127 sont communs à toutes les langues, mais les caractères 128 à 255 varient selon les langues. Faites attention si la langue de Unity Pro n'est pas la même que celle du SE. Si ces langues diffèrent, la communication CHAR MODE peut être perturbée et l'envoi de caractères supérieurs à 127 risque d'être incorrect. En particulier, si le caractère « Arrêt en réception » est supérieur à 127, il n'est pas pris en compte.

Règles de syntaxe

La saisie est précédée et terminée par le caractère apostrophe « ’ » (code ASCII 16#27).

Le signe $ (dollar) est un caractère spécial. Suivi de certaines lettres, il indique :
- $L ou $l, aller à la ligne suivante (line feed),
- $N ou $n, aller au début de la ligne suivante (new line),
- $P ou $p, aller à la page suivante (go to next page),
- $R ou $r, retour chariot (carriage return),
- $T ou $t, tabulation (Tab),
- $$, représente le caractère $ dans une chaîne,
- $’, représente le caractère apostrophe dans une chaîne.

L'utilisateur peut utiliser la syntaxe $nn pour afficher, dans une variable STRING, les caractères à ne pas imprimer. Il peut par exemple s'agir d'un retour chariot (code ASCII 16#0D).

Exemples

Exemples de saisies :

Type	**Entrée**	**Contenu de la chaîne** **• représente le caractère de fin de chaîne** *** représente les octets vides**
STRING	’ABCD’	ABCD•********** (16 caractères)
STRING[4]	jean•	jean•
STRING[10]	’It$’s jean’	It’s jean•*
STRING[5]	’’	•*****
STRING[5]	’$’’	’•****
STRING[5]	’le nombre’	le no•
STRING[13]	’0123456789’	0123456789•***
STRING[5]	’RL’	<cr><lf>•***
STRING[5]	’$$1.00’	$1.00•

Déclaration d’une variable de type STRING

Une variable de type STRING peut être déclarée de deux manières différentes :
- STRING et
- STRING[<Nombre d'éléments>]

Selon l'usage, le comportement varie :

Type	Déclaration d'une variable	Paramètre d'entrée d'un FFB	Paramètre de sortie d'un EF	Paramètre de sortie d'un FB
STRING	Taille fixe : 16 caractères	La taille est égale à la taille réelle du paramètre d'entrée.	La taille est égale à la taille réelle du paramètre d'entrée.	Taille fixe de 16 caractères
STRING[<n>]	Taille fixe : n caractères	La taille est égale à la taille réelle du paramètre d'entrée limitée à n caractères.	L'EF écrit un maximum de n caractères.	Le FB écrit un maximum de n caractères.

Les chaînes et la broche ANY

Lorsque vous utilisez une variable de type STRING comme paramètre de type ANY il est fortement recommandé de vérifier que la taille de la variable est inférieure à la taille maximum déclarée.

Exemple :

```
Utilisation de STRING sur la fonction SEL (Sélecteur).
String1 : STRING[8]
String2 : STRING[4]
String3 : STRING[4]

String1:= 'AAAAAAAA';
String3:= 'CC';
Cas 1 :
String2:= 'BBBB';
(* la taille de la chaîne est égale à la taille maximum
déclarée *)
String1:= SEL(FALSE, String2, String3);
(* le résultat sera : 'BBBBAAAA' *)
Cas 2 :
String2:= 'BBB';
(* la taille de la chaîne est inférieure à la taille maximum
déclarée*)
String1:= SEL(FALSE, String2, String3);
(* le résultat sera : 'BBB' *)
```

8.5 Types de données élémentaires (EDT) au format chaîne de bits

Objet de cette section

Cette section décrit le type de données au format chaîne de bits. qui sont :

- le type Byte,
- le type Word,
- le type Dword.

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Présentation des types de données au format chaîne de bits

Présentation

Les types de données au format chaine de bits appartiennent à la famille de données élémentaire EDT (Elementary data type) qui regroupe des types de données **dits simples** et non pas composées (tableaux, structure, bloc fonctions).

Rappel sur le format chaîne de bits

La particularité de ce format est que l'ensemble des bits qui le compose ne représente pas une valeur numérique, mais une combinaison de bits séparés.

Les données appartenant aux types de ce format peuvent être représentées sous trois bases qui sont :

- l'hexadécimal (16#),
- l'octal (8#),
- le binaire (2#).

Types de données au format chaîne de bits

Trois types de données:

Type	Echelle (bits)	Valeur par défaut
BYTE	8	0
WORD	16	0
DWORD	32	0

Types de chaînes de bits

Le type Byte

Le type Byte est codé sur un format de 8 bits.

Ce tableau donne les limites inférieure/supérieure des bases qui peuvent être utilisées.

Base	Limite inférieure	Limite supérieure
Hexadécimal	16#0	16#FF
Octal	8#0	8#377
Binaire	2#0	2#11111111

Exemples de représentation :

Donnée	Représentation dans l'une des bases
00001000	16#8
00110011	8#63
00110011	2#110011

Le type Word

Le type Word est codé sur un format de 16 bits.

Ce tableau donne les limites inférieure/supérieure des bases qui peuvent être utilisées.

Base	Limite inférieure	Limite supérieure
Hexadécimal	16#0	16#FFFF
Octal	8#0	8#177777
Binaire	2#0	2#1111111111111111

Exemples de représentation :

Donnée	Représentation dans l'une des bases
0000000011010011	16#D3
1010101010101010	8#125252
0000000011010011	2#11010011

Le type Dword

Le type Dword est codé sur un format de 32 bits.

Ce tableau donne les limites inférieure/supérieure des bases qui peuvent être utilisées.

Base	Limite inférieure	Limite supérieure
Hexadécimal	16#0	16#FFFFFFFF
Octal	8#0	8#37777777777
Binaire	2#0	2#11111111111111111111111111111111

Exemples de représentation :

Donnée	Représentation dans l'une des bases
00000000000010101101110011011110	16#ADCDE
00000000000000010000000000000000	8#200000
00000000000010101011110011011110	2#10101011110011011110

8.6 Types de données dérivés (DDT/IODDT)

Objet de cette section

Cette section décrit les types de données dérivés qui sont :

- les tableaux (DDT),
- d'EDT
  - les structures concernant les données d'entrées\sorties (IODDT),
  - les structures concernant les autres données (DDT).

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Tableaux

Qu'est-ce qu'un tableau ?

Un tableau est un élément de données incluant un **ensemble** de données **de type identique**, tel que :

- données élémentaires (EDT),
  par exemple :
  - un groupe de mots BOOL,
  - un groupe de mots entiers UINT,
  - etc.
- données dérivées (DDT),
  par exemple :
  - un groupe de tables WORD,
  - un groupe de structures,
  - etc.

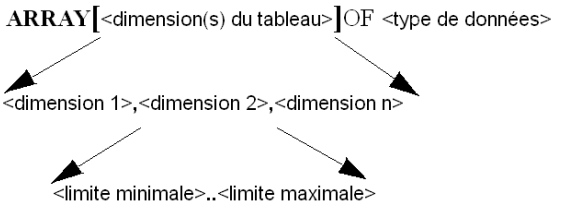
Caractéristiques

Un tableau est caractérisé par deux paramètres :

- un paramètre définissant sa présentation (dimension(s) du tableau),
- un paramètre définissant le type de données qu'il contient.

NOTE : la présentation la plus complexe est le tableau à **six dimensions**.

La syntaxe incluant ces deux paramètres se présente comme suit :



<limite minimale> **inférieure** à <limite maximale>

Définition et instanciation d'un tableau

Définition d'un type de tableau :

```
X : ARRAY[1.0,10] OF BOOL
```

Instanciation d'un tableau :

```
Tab_1: X
Tab_2: ARRAY[1.0,10] OF BOOL
```

Les instances Tab_1 et Tab_2 sont de même type et ont le même nombre de dimensions ; la seule différence concerne ce qui se passe lors de l'instanciation :

- Le type Tab_1 est nommé X.
- Le type Tab_2 doit être défini (la table correspondante ne porte pas de nom).

NOTE : il est préférable d'attribuer un nom au type, puisque chaque modification requise ne sera effectuée qu'une seule fois (en effet, dans le cas contraire, le nombre de modifications serait aussi important que le nombre d'instances).

Exemples

Le tableau ci-dessous répertorie des instances de tableaux dont le nombre de dimensions diffère :

Entrée	Commentaires
Tab_1: ARRAY[1..2] OF BOOL	Tableau à 1 dimension contenant 2 mots de type booléen
Tab_2: ARRAY[-10..20] OF WORD	Tableau à 1 dimension contenant 31 structures de type WORD (structure définie par l'utilisateur)
Tab_3: ARRAY[1..10, 1..20] OF INT	Tableaux à 2 dimensions contenant 10 x 20 nombres entiers
Tab_4: ARRAY[0..2, -1..1, 201..300, 0..1] OF REAL	Tableaux à 4 dimensions contenant 3 x 3 x 100 x 2 nombres réels

NOTE : un grand nombre de fonctions (READ_VAR, WRITE_VAR, par exemple) ne reconnaissent pas l'index d'un tableau de mots commençant par un nombre autre que 0. Si vous avez recours à ce type d'index, les fonctions examinent le nombre de mots du tableau, mais pas celui figurant au niveau du premier ensemble de l'index dans la définition du tableau.

⚠ AVERTISSEMENT

COMPORTEMENT INATTENDU DE L'APPLICATION - INDEX DE TABLEAU NON VALIDE

Lors de l'application de fonctions à des variables de type tableau, vérifiez que les fonctions sont compatibles avec la valeur de l'index de début des tableaux lorsque cette valeur est supérieure à 0.

Le non-respect de ces instructions peut provoquer la mort, des blessures graves ou des dommages matériels.

Accès à un élément de données dans le tableau Tab_1 ou Tab_3 :

```
Tab_1[2]
;Pour accéder au deuxième élément

Tab_3[4][18]
;Pour accéder au dix-huitième élément du quatrième sous-tableau
```

Règles d'affectation entre les tableaux

Les 4 tableaux suivants sont disponibles :

```
Tab_1:ARRAY[1..10] OF INT
Tab_2:ARRAY[1..10] OF INT
Tab_3:ARRAY[1..11] OF INT
Tab_4:ARRAY[101..110] OF INT


Tab_1: = Tab_2; affectation autorisée
Tab_1: = Tab_3; affectation refusée (non conforme CEI)
Tab_1: = Tab_4; affectation refusée (non conforme CEI)
```

Structures

Qu'est ce qu'une structure?

C'est une donnée qui contient un **ensemble** de données **de type différent** telles que:

- un ensemble de BOOL, WORD, UINT, etc., (structure EDT),
- un ensemble de tableaux (structure de DDT),
- un ensemble de REAL, DWORD, tableaux, etc..., (structure d'EDT et DDT).

NOTE : vous pouvez réaliser des structures imbriquées (DDT imbriqués) sur 8 niveaux. **Les structures (DDT) récursives ne sont pas autorisées.**

Caractéristiques

Une structure est constituée de données qui sont chacunes caractérisées par:

- un type,
- un nom, qui permet de l'identifier,
- un commentaire (optionnel) décrivant son rôle.

Définition d'un type de structure :

```
IDENT
    Nom : STRING[12]
    Prenom : STRING[16]
    Age : UINT

;La structure de type IDENT contient une donnée de type UINT et
deux données de types STRING
```

Définition de deux instances de donnée de la structure de type IDENT :

```
Personne_1 : IDENT
Personne_2 : IDENT

;Les instances Personne_1 et Personne_2 sont du type Structure
IDENT
```

Accés à une données d'une structure

Accès à des données de l'instance Personne_1 de type IDENT :

```
Personne_1.Nom  ;Permet d'acceder au nom de Personne_1

Personne_1.Age    ;Permet d'acceder à l'age de Personne_1
```

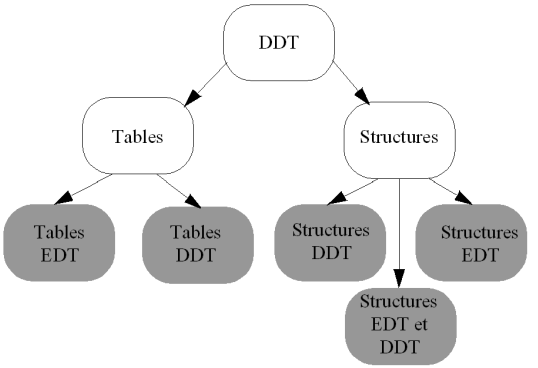
Vue d'ensemble de la famille de type de données dérivées (DDT)

Introduction

La famille DDT (Derived Data Type) inclut les types de données dit **"dérivés"** tels que :

- les tables ;
- les structures.

Illustration :



Caractéristiques

Un élément de données appartenant à la famille DDT comporte :

- le nom du type *(voir page 238)* (32 caractères maximum) défini par l'utilisateur (facultatif pour les tables, mais recommandé) *(voir page 272)* ;
- le type (structure ou table) ;
- un commentaire facultatif de 1 024 caractères maximum (les caractères autorisés correspondent aux codes ASCII 32 à 255) ;
- la description (dans le cas d'une structure) de ces éléments :
  - le nom de l'élément *(voir page 238)* (32 caractères maximum) ;
  - le type d'élément ;
  - un commentaire facultatif de 1 024 caractères maximum expliquant son rôle (les caractères autorisés correspondent aux codes ASCII 32 à 255).
- des informations :
  - le numéro de version du type ;
  - la date de la dernière modification du code, des variables internes ou des variables de l'interface ;
  - un fichier de 32 767 caractères décrivant la fonction bloc et ses différentes modifications (facultatif).

NOTE : La taille totale d'une table ou structure n'excède pas 64 Ko.

Exemples

Définition des types

```
COORD
    X : INT
    Y : INT
 ; Structure de type COORD

SEGMENT
    Origin: COORD
    Destination: COORD
 ; Structure de type SEGMENT contenant 2 structures de
type COORD

OUTLINE: ARRAY[0..99] OF SEGMENT
 ; Table de type OUTLINE contenant 100 structures de
type SEGMENT

DRAW
    Color : INT
    Anchor : COORD
    Pattern : ARRAY[0..15,0..15] OF WORD
    Contour : OUTLINE
 ; Structure de type DRAW
```

Accès aux données d'une instance de structure de type DRAW

```
Cartoon: DRAW
 ; Instance de structure de type DRAW

Cartoon.Pattern[15,15]
 ; Accès au dernier élément de données dans la table
Pattern de la structure Cartoon

Cartoon.Contour[0].Origin.X
 ; Accès à l'élément de données X de la structure COORD
appartenant à la première structure SEGMENT de la table
Contour.
```

DDT : règles d'affectation

Vue d'ensemble

Les DDT sont stockés dans la mémoire de l'automate selon l'ordre dans lequel leurs éléments sont déclarés.

Il existe cependant certaines règles que voici.

Principe pour Premium et Quantum

Le principe de stockage pour Premium et Quantum est le suivant :

- Les éléments sont stockés dans l'ordre selon lequel ils sont déclarés dans la structure.
- L'élément de base est l'octet (alignement des données sur les octets mémoire).
- Chaque élément possède une règle d'alignement :
  - Les types `BOOL` et `BYTE` sont indifféremment alignés sur les octets pairs ou impairs.
  - Tous les autres types élémentaires sont alignés sur les octets pairs.
  - Les structures et tableaux sont alignés selon la règle d'alignement des types `BOOL` et `BYTE` s'ils ne contiennent que des éléments de type `BOOL` et `BYTE` ; sinon, ils sont alignés sur les octets pairs de la mémoire.

⚠ AVERTISSEMENT

RISQUE D'INCOMPATIBILITE APRES LA CONVERSION DE CONCEPT

Avec l'application de programmation **Concept**, les structures de données ne traitent aucun décalage dans les offsets (chaque élément est défini l'un après l'autre dans la mémoire, quel que soit son type). Par conséquent, nous vous recommandons de tout vérifier, en particulier la cohérence des données lors de l'utilisation des `DDT` situés dans la « RAM d'état » (risque de décalages) ou des fonctions pour la communication avec d'autres équipements (transferts avec une taille différente de celle programmée dans Concept).

Le non-respect de ces instructions peut provoquer la mort, des blessures graves ou des dommages matériels.

Principe pour Modicon M340

Le principe de stockage pour les automates Modicon M340 est le suivant :

- Les éléments sont stockés dans l'ordre selon lequel ils sont déclarés dans la structure.
- L'élément de base est l'octet.
- Une règle d'alignement et une fonction de l'élément :
  - Les types `BOOL` et `BYTE` sont alignés sur les octets pairs ou impairs.
  - Les types `INT`, `WORD` et `UINT` sont alignés sur les octets pairs.
  - Les types `DINT`, `UDINT`, `REAL`, `TIME`, `DATE`, `TOD`, `DT` et `DWORD` sont alignés sur les mots doubles.
  - Les structures et les tables sont alignées selon les règles de leurs éléments.

⚠ AVERTISSEMENT

MAUVAIS ECHANGES ENTRE UN MODICON M340 ET UN PREMIUM OU UN QUANTUM.

Vérifiez si la structure des données échangées présente les mêmes alignements dans les deux projets.

Si tel n'est pas le cas, les données ne seront pas correctement échangées.

Le non-respect de ces instructions peut provoquer la mort, des blessures graves ou des dommages matériels.

NOTE : il est possible que l'alignement des données ne soit pas identique si le projet est transféré du simulateur de Unity Pro vers un automate M340. Vérifiez par conséquent la structure des données du projet.

NOTE : Unity Pro indique où l'alignement semble différent. Vérifiez les instances correspondantes dans l'éditeur de données. Voir la page Options du projet pour savoir comment activer cette option.

Exemples

Le tableau ci-dessous donne quelques exemples de structures de données. Dans ces exemples, les DDT de type structure sont adressés à `%MWi`. Le 1er octet du mot correspond aux 8 bits de poids faible et le 2e octet du mot correspond aux 8 bits de poids fort.

Pour toutes les structures suivantes, la première variable est affectée à l'adresse `%MW100` :

Première adresse mémoire		**Description de la structure**	
Modicon M340	Premium	**Para_PWM1**	
`%MW100` (1er octet)	`%MW100` (1er octet)		t_period : TIME

Première adresse mémoire		Description de la structure	
%MW102 (1er octet)	%MW102 (1er octet)		t_min : TIME
%MW104 (1er octet)	%MW104 (1er octet)		in_max : REAL
		Mode_TOTALIZER	
%MW100 (1er octet)	%MW100 (1er octet)		hold : BOOL
%MW100 (2e octet)	%MW100 (2e octet)		rst : BOOL
		Info_TOTALIZER	
%MW100 (1er octet)	%MW100 (1er octet)		outc : REAL
%MW102 (1er octet)	%MW102 (1er octet)		cter : UINT
%MW103 (1er octet)	%MW103 (1er octet)		done : BOOL
%MW103 (2e octet)	%MW103 (2e octet)		**Réservé pour l'alignement**

Le tableau ci-dessous donne deux exemples de structures de données avec tableaux :

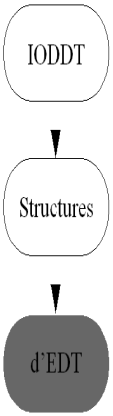
Première adresse mémoire		Description de la structure	
Modicon M340	Premium	**EHC105_Out**	
%MW100 (1er octet)	%MW100 (1er octet)		Quit : BYTE
%MW100 (2e octet)	%MW100 (2e octet)		Control : ARRAY [1.0,5] OF BYTE
%MW104 (1er octet)	%MW103 (1er octet)		Final : ARRAY [1..5] OF DINT
		CPCfg_ex	
%MW100 (1er octet)	%MW100 (1er octet)		Profile_type : INT
%MW101 (1er octet)	%MW101 (1er octet)		Interp_type : INT
%MW102 (1er octet)	%MW102 (1er octet)		Nb_of_coords : INT
%MW103 (1er octet)	%MW103 (1er octet)		Nb_of_points : INT
%MW104 (1er octet)	%MW104 (1er octet)		reserved : ARRAY [0..4] OF BYTE
%MW106 (2e octet)	%MW106 (2e octet)		**Réservé pour l'alignement de la variable Master_offset sur octets pairs**
%MW108 (1er octet)	%MW107 (1er octet)		Master_offset : DINT
%MW110 (1er octet)	%MW109 (1er octet)		Follower_offset : INT
%MW111 (mot entier)	-		**Réservé pour l'alignement**

Aperçu des types de données dérivés d'entrée/de sortie (IODDT)

Présentation

Les types de données dérivés d'entrées\sorties IODDT (Input Output Derived Data Type) **sont prédéfinis par le constructeur**, ils contiennent des objets langage de la famille EDT appartenant à la voie d'un module métier.

Illustration :



Les type IODDT sont des structures dont la taille (nombre d'éléments qui les composent) dépend de la voie ou du module d'entrées\sorties qu'elles représentent.

Un module d'entrées\de sorties donné peut avoir plus d'un IODDT.

La différence avec une structure classique est que:

- la structure IODDT est prédéfinie par le constructeur,
- les éléments composant la structure IODDT n'ont pas une allocation mémoire contigüe, ils ont une adresse spécifique dans le module.

Exemples

Structure IODDT pour une voie d'entrée\de sortie d'un module analogique

```
ANA_IN_GEN    ;Structure de type ANA_IN_GEN
   Value:INT ;Valeur de l'entrée
   Err:  BOOL ;Erreur voie
```

Accès à des données d'une instance de type ANA_IN_GEN :

```
 Cistern_Level: ANA_IN_GEN
;Instance de type ANA_IN_GEN qui correspond par exemple
à un capteur de niveau de cuve

Cistern_Level.Value ;Lecture valeur d'entrée de la voie
Cistern_Level.Err ;Lecture bit d'erreur de la voie
```

Accès par adressage direct :

```
S'il s'agit de la voie 0 du module 2 du rack 0 nous avons :

Cistern_Level       correspond à %CH0.2.0
Cistern_Level.Value correspond à %IW0.2.0.0
Cistern_Level_Err   correspond à %I0.2.0.ERR
```

8.7 Types de données blocs fonctions (DFB\EFB)

Objet de cette section

Cette section décrit les types de données de bloc fonction. qui sont :

- blocs fonction utilisateur (DFB) ;
- blocs fonction élémentaires (EFB).

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

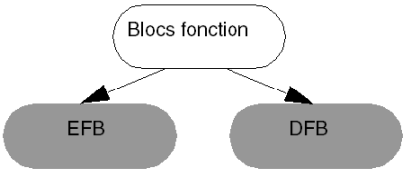
Vue d'ensemble des familles de type de données de bloc fonction

Présentation

Les familles de type de données blocs fonction sont:

- la famille de type Blocs fonction élémentaires (EFB) *(voir page 233)*,
- la famille de type Blocs fonction utilisateur (DFB) *(voir page 233)*.

Illustration :

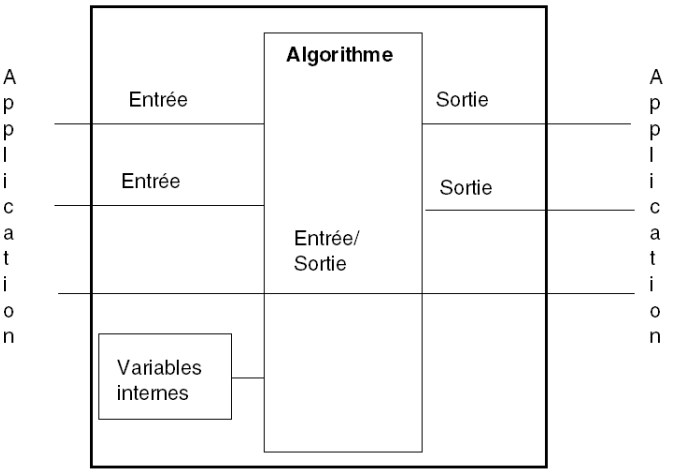


Les blocs fonctions sont des entités contenant:

- des variables d'entrées et de sorties servant d'interface avec l'application,
- un algorithme de traitement exploitant les variables d'entrées et renseignant les variables de sorties,
- des variables internes privées et publiques exploitées par l'algorithme de traitement.

Illustration

Bloc fonction :



Bloc fonction utilisateur (DFB)

Les types blocs fonction utilisateur (blocs fonction dérivés) sont développés par l'utilisateur avec un ou plusieurs langages (suivant le nombre de sections). Ces langages sont les suivants :

- à contacts,
- langage littéral structuré,
- langage liste d'instructions,
- langage en blocs fonctionnels FBD.

Un type DFB peut avoir une ou plusieurs instances, chaque instance est référencée par un nom (symbole) et possède les données du type de DFB.

les blocs fonction élémentaires (EFB)

Les types blocs fonctions élémentaires (EFB) sont fournis par le constructeur, ils sont programmés en langage C.

L'utilisateur peut créer ses EFB, pour cela il doit disposer d'un outil logiciel optionnel **"SDKC"**

Un type EFB peut avoir une ou plusieurs instances, chaque instance est référencée par un nom (symbole) et possède les données du type de l'EFB.

Caractéristiques des types de données de bloc fonction (EFB\DFB)

Définition du type

Le type d'un bloc fonction EFB ou DFB est défini par :

- le nom du type *(voir page 238)*, défini par l'utilisateur pour les DFB,
- un commentaire facultatif. (les caractères autorisés correspondent aux codes ASCII 32 à 255) ;
- les données d'interface avec l'application:
  - les entrées, pas accessibles en lecture\écriture depuis l'application, mais lues par le code du bloc fonction,
  - les entrées\sorties, pas accessibles en lecture\écriture depuis l'application, mais lues et écrites par le code du bloc fonction,
  - les sorties, accessibles en lecture depuis l'application et lues et écrites par le code du bloc fonction.
- les donnéees internes:
  - publiques, accessibles en lecture\écriture depuis l'application, lues et ecrites par le code du bloc fonction,
  - privées, pas accessibles depuis l'application lues et écrites par le code du bloc fonction.
- le code:
  - pour les DFB il est écrit par l'utilisateur en langage d'automatisme (littéral structuré, liste d'instructions, langage à contacts, langage à blocs fonctionnels), il est structuré en une seule section si l'option IEC est active, ou peut être structuré en plusieurs sections si cette option est inactive
  - pour les EFB il est écrit en langage C.
- des informations telles que:
  - le numéro de version du type,
  - la date de la derniére modification du code, ou des variables internes, ou des variables interfaces.
  - une fiche descriptive facultative (32767 caractères), décrivant la fonction du bloc et ses différentes modifications.

Caractéristiques

Ce tableau donne les caractéristiques des éléments composant un type:

Elément	EFB	DFB
Nom	32 caractères	32 caractères
Commentaire	1024 caractères	1024 caractères
Données d'Entrées	32 maximum	32 maximum
Données d'Entrées/Sorties	32 maximum	32 maximum
Données de sortie	32 maximum	32 maximum
Nombre d'interfaces (Entrées+Sorties+Entrées/Sorties)	32 maximum **(2)**	32 maximum **(2)**
Données publiques	Pas de limites **(1)**	Pas de limites **(1)**
Données privées	Pas de limites **(1)**	Pas de limites **(1)**
Langage de programmation	Langage C	Langage: ● littéral structuré, ● liste d'instructions, ● à contacts, ● à blocs fonctionnels.

Section		Une section est définie par:
		● un nom (32 caractères maximum),
		● une condition de validation,
		● un commentaire (256 caractéres maximum),
		● une protection:
		  ● sans,
		  ● lecture,
		  ● lecture\écriture.
		Une section ne peut pas accéder aux variables déclarées dans l'application sauf les:
		● doubles mots systéme %SDi,
		● mots système %SWi,
		● bits système %Si.

(1): la seule limitation est la taille mémoire de l'automate.

(2): ne sont pas prises en compte l'entrée EN et la sortie ENO.

Caractéristiques d'éléments appartenant aux blocs fonction

Qu'est-ce qu'un élément ?

Chaque élément (données d'interface ou données internes) est défini par :

- un nom *(voir page 238)* (comportant 32 caractères maximum), attribué par l'utilisateur ;
- un type,
  qui fait partie de l'une des familles ci-après :
  - données élémentaires (EDT) ;
  - données dérivées (DDT) ;
  - données de blocs fonction (EFB/DFB).
- un commentaire facultatif de 1 024 caractères maximum (les caractères autorisés correspondent aux codes ASCII 32 à 255) ;
- une valeur initiale ;
- un droit d'accès issu du programme d'application (pour afficher des sections de l'application ou la section appartenant aux DFB, reportez-vous à la rubrique Définition du type de bloc fonction (variables d'interface et internes) *(voir page 285)*) ;
- un droit d'accès provenant des requêtes de communication ;
- un indicateur de sauvegarde de variables publiques.

Types de données autorisés pour un élément appartenant à un DFB

Les types de données autorisés sont indiqués dans le tableau ci-dessous :

Elément du DFB	**Types EDT**	**Types DDT**				**ANY...**	**Types de bloc fonction**
		IODDT	Tables sans nom	ANY_ARRAY	autre		
Données d'entrée	Oui	Non	Oui	Oui	Oui	Oui**(2)**	Non
Données d'entrée/de sortie	Oui**(1)**	Oui	Oui	Oui	Oui	Oui**(2)**	Non
Données de sortie	Oui	Non	Oui	Non	Oui	Oui **(2) (3)**	Non
Données publiques	Oui	Non	Oui	Non	Oui	Non	Non
Données privées	Oui	Non	Oui	Non	Oui	Non	Oui

(1) : non autorisé pour les données statiques de type EBOOL utilisées sur les automates Quantum

(2) : non autorisé pour les données de type BOOL et EBOOL

(3) : achèvement lors de l'exécution du DFB et utilisation impossible en dehors du DFB

Types de données autorisés pour un élément appartenant à un EFB

Les types de données autorisés sont indiqués dans le tableau ci-dessous :

Elément de l'EFB	Types EDT	Types DDT				ANY...	Types de bloc fonction
		IODDT	Tables sans nom	ANY_ARRAY	autre		
Données d'entrée	Oui	Non	Non	Oui	Oui	Oui**(1)**	Non
Données d'entrée/de sortie	Oui	Oui	Non	Oui	Oui	Oui**(1)**	Non
Données de sortie	Oui	Non	Non	Non	Oui	Oui **(1) (2)**	Non
Données publiques	Oui	Non	Non	Non	Oui	Non	Non
Données privées	Oui	Non	Non	Non	Oui	Non	Oui

(1) : non autorisé pour les données de type BOOL et EBOOL

(2) : achèvement lors de l'exécution de l'EFB et utilisation impossible en dehors de l'EFB

Valeurs initiales pour un élément appartenant à un DFB

Le tableau suivant précise si les valeurs initiales entrées sont issues de la définition du type DFB ou de l'instance DFB :

Elément du DFB	Provenant du type DFB	Provenant de l'instance EFB
Données d'entrée (pas de type ANY...)	Oui	Oui
Données d'entrée (de type ANY...)	Non	Non
Données d'entrée/de sortie	Non	Non
Données de sortie (pas de type ANY...)	Oui	Oui
Données de sortie (de type ANY...)	Non	Non
Données publiques	Oui	Oui
Données privées	Oui	Non

Valeurs initiales pour un élément appartenant à un EFB

Le tableau suivant précise si les valeurs initiales entrées sont issues de la définition du type EFB ou de l'instance EFB :

Elément de l'EFB	Provenant du type EFB	Provenant de l'instance EFB
Données d'entrée (pas de type ANY..., voir generic data types *(voir page 290)*)	Oui	Oui
Données d'entrée (de type ANY...)	Non	Non
Données d'entrée/de sortie	Non	Non
Données de sortie (pas de type ANY...)	Oui	Oui
Données de sortie (de type ANY...)	Non	Non
Données publiques	Oui	Oui
Données privées	Oui	Non

⚠ AVERTISSEMENT

COMPORTEMENT INATTENDU DE L'APPLICATION - INDEX DE TABLEAU NON VALIDE

Lorsque des EFB et des DFB sont utilisés sur des variables de type tableau, faites uniquement appel à des tableaux avec un index de début égal à 0.

Le non-respect de ces instructions peut provoquer la mort, des blessures graves ou des dommages matériels.

8.8 Types de données génériques (GDT)

Vue d'ensemble des types de données génériques

Présentation

Les types de données génériques sont des ensembles de types classiques (EDT, DDT) ayant pour vocation de déterminer la compatibilité entre ces types classiques.

Ces ensembles sont identifiés par le préfixe ANY_ARRAY, mais ces préfixes ne peuvent en aucun cas être utilisés pour instancier des données.

Leurs champs d'utilisation concernent les familles de type de données blocs fonctions (EFB\DFB) et les fonctions élémentaires (EF), afin de définir les types de données compatibles avec leur interface :

- entrées
- entrées/sorties
- sorties

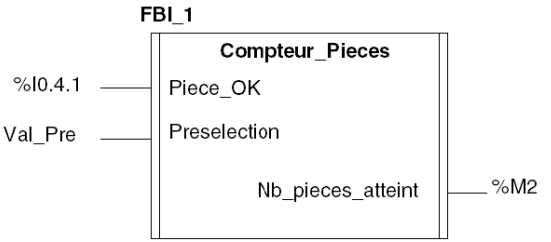
Types de données génériques (GDT) disponibles

Les types de données génériques disponibles dans Unity Pro sont les suivants :

- ANY_ARRAY_WORD
- ANY_ARRAY_UINT
- ANY_ARRAY_UDINT
- ANY_ARRAY_TOD
- ANY_ARRAY_TIME
- ANY_ARRAY_STRING
- ANY_ARRAY_REAL
- ANY_ARRAY_INT
- ANY_ARRAY_EBOOL
- ANY_ARRAY_DWORD
- ANY_ARRAY_DT
- ANY_ARRAY_DINT
- ANY_ARRAY_DATE
- ANY_ARRAY_BYTE
- ANY_ARRAY_BOOL

Exemple

Soit le DFB suivant :



Le paramètre d'entrée **Présélection** peut être défini de type GDT.

NOTE : Les objets autorisés pour les différents paramètres sont définis dans ce tableau *(voir page 573)*.

8.9 Types de données appartenant aux diagrammes fonctionnels en séquence (SFC)

Vue d'ensemble des types de données de la famille du diagramme fonctionnel en séquence

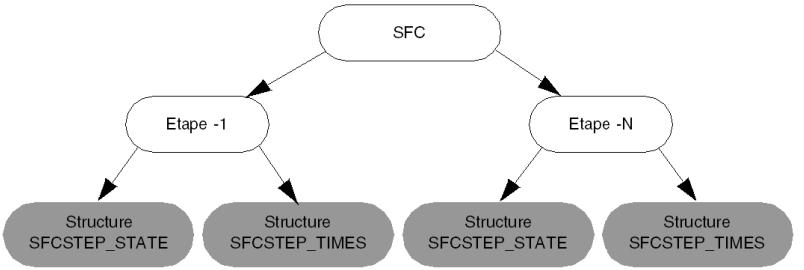
Présentation

La famille des types de données diagrammes fonctionnels en séquence SFC (Sequential function chart) regroupe des types de données **dits composés** tels que des structures restituant les propriétés et l'état du graphe (Chart) et des actions le composant.

Chaque étape est représentée par deux structures qui sont :

- la structure **SFCSTEP_STATE**,
- la structure **SFCSTEP_TIMES**.

Illustration :



NOTE : Les deux types de structures **SFCSTEP_STATE** et **SFCSTEP_TIMES** sont rattachées aussi à chaque Macro-étape du diagramme fonctionnel en séquence.

Définition de la structure de type SFCSTEP_STATE

Cette structure rassemble toutes les données liées à l'état de l'étape ou de la Macro-étape.

Ces données sont:

- **x**: donnée élémentaire (EDT) de type BOOL contenant la valeur TRUE quand l'étape est active,
- **t**: donnée élémentaire (EDT) de type TIME contenant le temps d'activité de l'étape. Lorsqu'il est désactivé, la valeur de l'étape est maintenue jusqu'à la prochaine activation,
- **tminErr**: donnée élémentaire (EDT) de type BOOL contenant la valeur TRUE si le temps d'activité de l'étape est inférieur au temps d'activité minimal programmé,
- **tmaxErr**: donnée élémentaire (EDT) de type BOOL contenant la valeur TRUE si le temps d'activité de l'étape est supérieur au temps d'activité maximal programmé,

Ces données sont accessibles à partir de l'application en lecture seule.

Définition de la structure de type SFCSTEP_TIMES

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

Ces données sont:

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

Ces données sont accessibles uniquement à partir de l'éditeur du SFC.

Syntaxe d'accès à des données de la structure SFCSTEP_STATE

Les noms d'instances de cette structure correspondent aux noms des étapes ou macro-étapes du diagramme fonctionnel en séquence

Syntaxe	Commentaire
Nom_Etape.x	Permet de connaitre l'état de l'étape (active\inactive)
Nom_Etape.t	Permet de connaitre le temps d'activation en cours ou total de l'étape
Nom_Etape.tminErr	Permet de connaitre si le temps minimal d'activation de l'étape est inférieur au temps programmé dans Nom-Etape.tmin
Nom_Etape.tmaxErr	Permet de connaitre si le temps maximal d'activation de l'étape est supérieur au temps programmé dans Nom-Etape.tmax

8.10 Compatibilité entre types de données

Compatibilité entre des types de données

Présentation

Ci-dessous sont présentées les différentes règles de compatibilité entre types **à l'intérieur** de chaque famille suivantes :

- la famille Type de données élémentaires (EDT),
- la famille Type de données dérivées (DDT),
- la famille Type de données génériques (GDT).

Famille Type de données élémentaires (EDT)

La famille de types de données élémentaires (EDT) contient ses sous-familles qui sont :

- la sous-famille de types de données au format binaire,
- la sous-famille de types de données au format BCD,
- la sous-famille de types de données au format Réel,
- la sous-famille de types de données au format chaîne de caractères,
- la sous-famille de types de données au format chaîne de bits.

Il n'y a pas de compatibilité entre deux types de données quels qu'ils soient, même s'ils appartiennent à la même sous-famille.

Famille Type de données dérivées (DDT)

La famille de types de données dérivées (DDT) contient ses sous-familles qui sont :

- la sous-famille de type tableaux,
- la sous-famille de type structures :
  - les structures concernant les données d'entrées\sorties (IODDT),
  - structures concernant les autres données.

Règles concernant les structures :

Deux structures sont compatibles si leurs éléments sont :

- de même nom,
- de même type,
- organisés suivant le même ordre.

Soit quatre types de structures :

```
ELEMENT_1
  My_Element : INT
  Other_Element : BOOL
; Structure de type ELEMENT_1


ELEMENT_2
  My_Element: INT
  Other_Element : BOOL
; Structure de type ELEMENT_2

ELEMENT_3
  Element : INT
  Other_Element : BOOL
; Structure de type ELEMENT_3

ELEMENT_4
  Other_Element : BOOL
  My_Element : INT
; Structure de type ELEMENT_4
```

Compatibilité entre les types de structures

Types	**ELEMENT_1**	**ELEMENT_2**	**ELEMENT_3**	**ELEMENT_4**
ELEMENT_1		OUI	NON	NON
ELEMENT_2	OUI		NON	NON
ELEMENT_3	NON	NON		NON
ELEMENT_4	NON	NON	NON	

Règles concernant les tableaux

Deux tableaux sont compatibles si :

- leurs dimensions et l'organisation de leurs dimensions sont identiques,
- chaque dimension correspondante est de même type.

Soit cinq types de tableaux :

```
TAB_1 : ARRAY[10..20]OF INT
; Tableau une dimension de type TAB_1

TAB_2: ARRAY[20..30]OF INT
; Tableau une dimension de type TAB_2

TAB_3: ARRAY[20..30]OF INT
; Tableau une dimension de type TAB_3

TAB_4: ARRAY[20..30]OF TAB_1
; Tableau une dimension de type TAB_4

TAB_5: ARRAY[20..30,10..20]OF INT
; Tableau deux dimensions de type TAB_5
```

Compatibilité entre les types de tableaux:

Le type...	**et le type...**	**sont...**
TAB_1	TAB_2	incompatibles
TAB_2	TAB_3	compatibles
TAB_4	TAB_5	compatibles
TAB_4[25]	TAB_5[28]	compatibles

Famille Type de données génériques (GDT)

La famille de types de données génériques (GDT) est composée d'ensembles organisés de façon hiérarchique qui contiennent des types de données appartenant aux familles :

- de types de données élémentaires (EDT),
- de types de données dérivées (DDT).

Règles :

Un type de données classique est compatible avec les types de données génériques qui lui sont hiérarchiques.

Un type de données génériques est compatible avec les types de données génériques qui lui sont hiérarchiques.

Exemple :

```
Le type INT est compatible avec les types ANY_INT ou ANY_NUM ou
ANY_MAGNITUDE.

Le type INT n'est pas compatible avec le type ANY_BIT ou ANY_REAL.

Le type générique ANY_INT est compatible avec le type ANY_NUM.

Le type générique ANY_INT n'est pas compatible avec le type
ANY_REAL.
```

Instances de données

9

Contenu du chapitre

Ce chapitre décrit les instances de données et leurs caractéristiques.

Ces instances peuvent être:

- des instances de données non localisées,
- des instances de données localisées,
- des instances de données à adressage direct.

Contenu de ce chapitre

Ce chapitre contient les sujets suivants :

Instances de types de données

Introduction

Qu'est ce qu'une instance de types de données ? *(voir page 235)*

L'instance d'un type de données est référencée soit par :

- **un nom (symbole)**, dans ce cas on dit que les données sont **non localisées**, car leur allocation mémoire n'est pas définie mais effectuée automatiquement par le système,
- **un nom (symbole)** et **une adresse topologique** définie par le constructeur, dans ce cas on dit que les données sont **localisées**, car leur allocation mémoire est connue,
- **une adresse topologique** définie par le constructeur, dans ce cas on dit que la donnée est à **adressage direct**, son allocation mémoire est connue.

Instances de données non localisées

Les instances de données non localisées sont gérées par le système d'exploitation de l'automate, leur emplacement physique dans la mémoire n'est pas connu de l'utilisateur.

Les instances de données non localisées sont définies à partir de types de données pouvant appartenir à la famille :

- de types de données élémentaires (EDT),
- de types de données dérivés (DDT).
- de types de données de blocs fonction (EFB/DFB),
- de type de données diagramme fonctionnel en séquence (SFC).

Exemples :

```
Var_1 : BOOL
;Instance de la famille EDT de type booléen allocation mémoire 1 octet

Var_2 : UDINT
;Instance de le famille EDT de type entier double non signé allocation
mémoire 4 octets

Var_3 : ARRAY[1..10]OF INT
;Instance de la famille DDT de type tableau allocation mémoire 20 octets

COORD
   X : INT
   Y : INT
Var_4 : COORD
;Instance de la famille DDT d'une structure de type COORD allocation
mémoire 4 octets
```

NOTE : Les instances de types de données de diagramme fonctionnel en séquence (SFC) sont créées au moment où elles sont insérées dans le programme d'application avec un nom par défaut que l'utilisateur peut modifier.

Instances de données localisées

La localisation d'une variable (définie par un symbole) consiste à créer une adresse dans l'éditeur de variables.

Les instances de données localisées possèdent un emplacement mémoire prédéfini dans l'automate et cet emplacement est connu de l'utilisateur.

- Adresse topologique pour les modules d'entrée/sortie
- Adresse globale (M340, Premium) ou RAM d'état (Quantum)

Les instances de données localisées sont définies à partir de types de données pouvant appartenir à la famille :

- de types de données élémentaires (EDT),
- de types de données dérivés (DDT),
- de types de données dérivés d'entrée/sortie (IODDT).

La liste ci-dessous présente les instances de données à localiser sur un type d'adresse %MW, %KW :

- INT,
- UINT,
- WORD,
- BYTE,
- DATE,
- DT,
- STRING,
- TIME,
- TOD,
- type de structure DDT,
- table.

Les tables EBOOL ou EBOOL et les instances de données doivent être localisées sur un type d'adresse %M, %Q ou %I.

Le type d'instance de données IODDT doit être localisé par le type de voie de module %CH.

NOTE : Les instances de type double de données localisées (DINT, DUNIT, DWORD) ou les instances flottantes (REAL) doivent être localisées par le type d'adresse %MW, %KW. Seule la localisation du type d'instance objets d'E/S est possible avec le type %MD<i>, %KD<i>, %QD, %ID, %MF<i>, %KF<i>, %QF, %IF en utilisant leur adresse topologique (exemple : %MD0.6.0.11, %MF0.6.0.31).

NOTE : Pour Modicon M340, la valeur (i) d'index doit être paire *(voir page 278)* pour les instances de données localisées de type double (%MW et %KW).

Exemples :

```
Var_1 : EBOOL AT %M100
;Instance de la famille EDT de type booléen (allocation mémoire 1 octet)
prédéfinie en %M100

Var_2 : BOOL AT %I2.1.0.ERR
;Instance de la famille EDT de type booléen (allocation mémoire 1 octet)
prédéfinie en %I2.1.0.ERR

Var_3 : INT AT %MW10
;Instance de la famille EDT de type entier (allocation mémoire 2 octets)
prédéfinie en %MW10

Var_4 : : DINT AT %MW1
;Interdit pour Modicon M340. Les instances de données localisées de type
double doivent avoir un événement d'adresse topologique (%MW2, %MW10.....).

Var_5 : WORD AT %MW10
;Instance de la famille EDT de type WORD (allocation mémoire 2 octets)
prédéfinie en %MW10

Var_6 : ARRAY[1..10]OF INT AT %MW50
;Instance de la famille DDT de type tableau (allocation mémoire 20 octets)
prédéfinie à partir de %MW50

COORD
   X : INT
   Y : INT

Var_7 : COORD AT %MW20
;Instance de la famille DDT d'une structure de type COORD (allocation mémoire
4 octets) prédéfinie à partir de %MW20

Var_8 : DINT AT %MD0.6.0.11
;Instance de la famille EDT de type DINT (allocation mémoire 4 octets)
prédéfinie à partir de l'adresse topologique de l'objet E/S du module métier


Var_9 : REAL AT %MF0.6.0.31
;Instance de la famille EDT de type REAL (allocation mémoire 4 octets)
prédéfinie à partir de l'adresse topologique de l'objet E/S du module métier
```

NOTE : Les instances de types de données de diagramme fonctionnel en séquence (SFC) sont créées au moment où elles sont insérées dans le programme d'application avec un nom par défaut que l'utilisateur peut modifier.

Instances de données à adressage direct

Les instances de données à adressage direct possèdent un emplacement prédéfini dans la mémoire automate ou dans un module métier et cet emplacement est connu de l'utilisateur.

Les instances de données à adressage direct sont définies à partir de types appartenant à la famille de type de données élémentaires (EDT).

Exemples d'instances de données à adressage direct :

Interne	Constantes	Système	Entrées/Sorties	Réseau
%Mi		**%Si**	**%Q, %I**	
%MWi	**%KWi**	**%SWi**	**%QW, %IW**	**%NW**
%MDi (1)	**%KDi** (1)		**%QD, %ID**	
%MFi (1)	**%KFi** (1)		**%QF, %IF**	
Légende				
(1) Non disponible pour Modicon M340				

NOTE : les instances de données localisées peuvent être utilisées par un adressage direct dans le programme.

Exemple :

- Var_1 : DINT AT `%MW10`
  ;%MW10 et %MW11 sont utilisés tous les deux. L'adressage direct `%MD10` peut être utilisé ou Var_1 dans le programme.

Attributs des instances de données

Présentation

Les attributs d'une instance de données sont des informations qui la définissent.

Ces informations sont:

- son nom *(voir page 238)* (sauf pour les instances de données à adressage direct *(voir page 306)*),
- son adresse topologique (sauf pour les instances de types de données non localisées),
- son type de données qui peut appartenir à la famille:
  - de type de données élémentaire (EDT),
  - de type de données dérivé (DDT),
  - de type de données blocs fonctions (EFB\DFB),
  - de type de données de diagramme fonctionnel séquentiel (SFC).
- un commentaire descriptif optionnel (de 1 024 caractères au maximum). Les caractères autorisés correspondent aux codes ASCII 32 à 255

Nom d'une instance de données

Il s'agit du symbole (32 caractères maximum) choisi par l'utilisateur qui permet de référencer l'instance, il doit être unique.

Certains noms ne peuvent être utilisés, par exemple:

- des mots-clés utilisés dans les langages textuels,
- des noms de section de programme,
- des noms de types de données prédéfinis ou choisis par l''utilisateur (structures, tableaux),
- des noms de types de DFB\EFB prédéfinis ou choisis par l'utilisateur,
- des noms de fonctions élémentaires (EF) prédéfinis ou choisis par l'utilisateur.

Nom d'instances appartenant à la famille SFC

Les noms des instances sont déclarés implicitement pendant que l'utilisateur dessine son diagramme fonctionnel en séquences. Ce sont des noms par défaut fournis par le constructeur que l'utilisateur peut modifier.

Noms par défaut fournis par le constructeur:

Objet SFC	Nom
Etape	S_<nom section>_<n° d'étape>
Etape de Macro-Etape	S_<nom section>_<n° macro-etape>_<n° d'étape>
Macro-étape	MS_<nom section>_<n° d'étape>
Macro-Etape imbriquée	MS_<nom section>_<n° macro-etape>_<n° d'etape>
Etape d'entrée de Macro-Etape	S_IN<nom de section>_<n° macro etape>

Objet SFC	Nom
Etape de sortie de Macro-Etape	S_OUT<nom de section>_<n° macro etape>
Transition	T_<nom section>_<n° transition>
Transition de Macro-Etape	T_<nom section>_<n° macro etape>_<n° transition>

Nom d'instances appartenant à la famille blocs fonctions

Les noms des instances sont déclarés implicitement pendant que l'utilisateur insère les instances dans les sections du programme d'application. Ce sont des noms par défaut fournis par le constructeur que **l'utilisateur peut modifier**.

Syntaxe des noms par défaut fournis par le constructeur :

```
FB_<nom du type de bloc fonction>_<n°d'instance>
```

NOTE : Le nom d'instance n'inclut pas le nom de section dans laquelle l'instance est utilisée, car elle peut être utilisée dans différentes sections de l'application.

Accès à un élément d'une instance de la famille DDT

La syntaxe d'accès est la suivante :

```
Cas des données de types structures
<Nom de l'instance>.<Nom de l'élément>

Cas des données de types tableaux
<Nom de l'instance>[Index de l'élément]
```

Régle:

La taille maximum de la syntaxe d'accès est de 1024 caractères maximum, et les limites possibles d'un type de données dérivé sont les suivantes:

- 10 niveaux d'imbrication (tableaux\structures),
- 6 dimensions par tableau,
- 4 chiffres pour définir l'index de l'élément d'un tableau.

Instances de données à adressage direct

Présentation

Qu'est ce qu'une instance de données à adressage direct ? *(voir page 303)*

Syntaxe d'accès

La syntaxe d'une instance de données à adressage direct est définie par le symbole **%** suivi d'un **préfixe de localisation mémoire** et dans certains cas d'informations supplémentaires.

Le préfixe de localisation mémoire peut être:

- **M**, pour les variables internes,
- **K**, pour les constantes (Premium et Modicon M340),
- **S**, pour les variables systèmes,
- **N**, pour les variables réseaux,
- **I**, pour les variables d'entrées,
- **Q**, pour les variables de sorties.

Variables internes %M

Syntaxe d'accès :

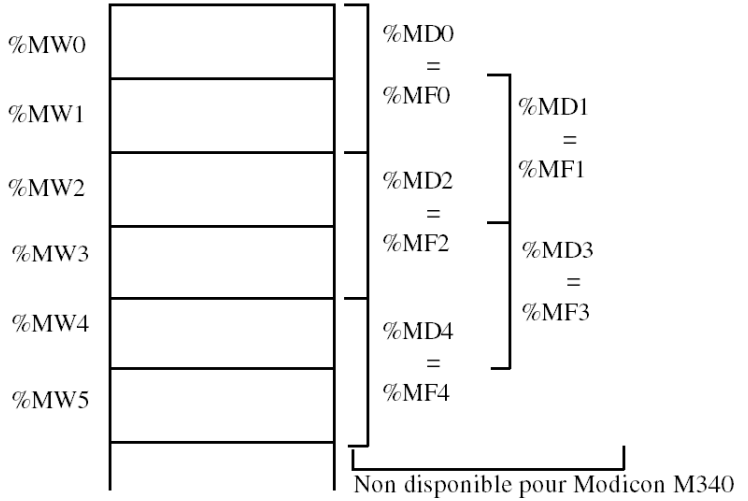
	Syntaxe	**Format**	**Exemple**	**Droit d'accès programme**
Bit	%M<i> ou %MX<i>	3 bits (EBOOL)	%M1	R/W
Mot	%MW<i>	16 bits (INT)	%MW10	R/W
Bit extrait de Mot	%MW<i>.<j>	1 bit (BOOL)	%MW15.5	R/W
Double mot	%MD<i> (1)	32 bits (DINT)	%MD8	R/W
Réel (flottant)	%MF<i> (1)	32 bits (REAL)	%MF15	R/W
Légende				
(1): Non disponible pour Modicon M340.				

<i> représente le numéro de l'instance (commence à 0 pour Premium et à 1 pour Quatum).

Pour le Modicon M340, les instances de type double (mot double) ou de flottement (réelles) doivent se situer dans un type d'entier %MW. L'index <i> de %MW doit être pair.

NOTE : Les données %M<i> ou %MX<i> détectent les fronts et gérent le forçage.

Réorganisation de la mémoire :



NOTE : La modification de %MW<i> entraîne les modifications de %MD<i> et %MF<i> correspondants.

Constantes %K

Syntaxe d'accès :

	Syntaxe	**Format**	**Droit d'accès programme**
Constante Mot	%KW<i>	16 bits (INT)	R
Constante double mot	%KD<i> (1)	32 bits (DINT)	R
Constante Réel (flottant)	%KF<i> (1)	32 bits (REAL)	R
Légende			
(1): Non disponible pour Modicon M340.			

<i> représente le numéro de l'instance.

NOTE : L'organisation de la mémoire est identique à celle des variables internes. Ces variables ne sont pas disponibles sur les automates Quantum.

Constantes %I

Syntaxe d'accès :

	Syntaxe	Format	Droit d'accès programme
Constante Bit	%I<i>	3 bits (EBOOL)	R
Constante Mot	%IW<i>	16 bits (INT)	R

<i> représente le numéro de l'instance.

NOTE : Ces données sont uniquement disponibles sur les automates Quantum et Momentum.

Variables système %S

Syntaxe d'accès :

	Syntaxe	Format	Droit d'accès programme
Bit	%S<i> ou %SX<i>	1 bit (BOOL)	R/W ou R
Mot	%SW<i>	32 bits (INT)	R/W ou R

<i> représente le numéro de l'instance.

NOTE : L'organisation de la mémoire est identique à celle des variables internes. Les données %S<i> et %SX<i> ne sont pas utilisées pour détecter les fronts et ne gèrent pas le forçage.

Variables réseaux %N

Ces variables contiennent des informations devant être échangées entre plusieurs programmes applicatifs à travers le réseau de communication.

Syntaxe d'accès :

	Syntaxe	Format	Droit d'accès programme
Mot commun	%NW<n>.<s>.<d>	16 bits (INT)	R\W ou R
Bit extrait de Mot	%NW<n>.<s>.<d>.<j>	1 bit (BOOL)	R\W ou R

<n> représente le numéro de réseau.

<s> représente le numéro de la station.

<d> représente le numéro de la donnée.

<j> représente le rang du bit dans le mot.

Cas des variables d'entrées/sorties

Ces variables sont contenues dans les modules métiers.

Syntaxe d'accès:

	Syntaxe	**Exemple**	**Droit d'accès programme**
Structure d'entrées\sortie (IODDT)	%CH<@mod>.<c>	%CH4.3.2	R
Entrées %I			
Bit d'erreur module de type BOOL	%I<@mod>.MOD.ERR	%I4.2.MOD.ERR	R
Bit d'erreur voie de type BOOL	%I<@mod>.<c>.ERR	%I4.2.3.ERR	R
Bit de type BOOL ou EBOOL	%I<@mod>.<c>	%I4.2.3	R
	%I<@mod>.<c>.<d>	%I4.2.3.1	R
Mot de type INT	%IW<@mod>.<c>	%IW4.2.3	R
	%IW<@mod>.<c>.<d>	%IW4.2.3.1	R
Double mot de type DINT	%ID<@mod>.<c>	%ID4.2.3	R
	%ID<@mod>.<c>.<d>	%ID4.2.3.2	R
Réel (flottant) de type REAL	%IF<@mod>.<c>	%IF4.2.3	R
	%IF<@mod>.<c>.<d>	%IF4.2.3.2	R
Sorties %Q			
Bit de type EBOOL	%Q<@mod>.<c>	%Q4.2.3	R/W
	%Q<@mod>.<c>.<d>	%Q4.2.3.1	R/W
Mot de type INT	%QW<@mod>.<c>	%QW4.2.3	R/W
	%QW<@mod>.<c>.<d>	%QW4.2.3.1	R/W
Double mot de type DINT	%QD<@mod>.<c>	%QD4.2.3	R/W
	%QD<@mod>.<c>.<d>	%QD4.2.3.2	R/W
Réel (flottant) de type REAL	%QF<@mod>.<c>	%QF4.2.3	R/W
	%QF<@mod>.<c>.<d>	%QF4.2.3.2	R/W
Variables %M (Premium)			
Mot de type INT	%MW<@mod>.<c>	%MW4.2.3	R/W
	%MW<@mod>.<c>.<d>	%MW4.2.3.1	R/W
Double mot de type DINT	%MD<@mod>.<c>	%MD4.2.3	R/W
	%MD<@mod>.<c>.<d>	%MD4.2.3.2	R/W
Réel (flottant) de type REAL	%MF<@mod>.<c>	%MF4.2.3	R/W
	%MF<@mod>.<c>.<d>	%MF4.2.3.2	R/W

	Syntaxe	Exemple	Droit d'accès programme
Constantes %K (Modicon M340 et Premium)			
Mot de type INT	%KW<@mod>.<c>	%KW4.2.3	R
	%KW<@mod>.<c>.<d>	%KW4.2.3.1	R
Double mot de type DINT	%KD<@mod>.<c>	%KD4.2.3	R
	%KD<@mod>.<c>.<d>	%KD4.2.3.12	R
Réel (flottant) de type REAL	%KF<@mod>.<c>	%KF4.2.3	R
	%KF<@mod>.<c>.<d>	%KF4.2.3.12	R

<@mod = \<b>.<e>\<r>.<m>

<b> numéro de bus (omis si station locale).

<e> numéro du point de connexion de l'équipement (omis si station locale, le point de connexion est appelé aussi Drop pour les utilisateurs d'automates Quantum).

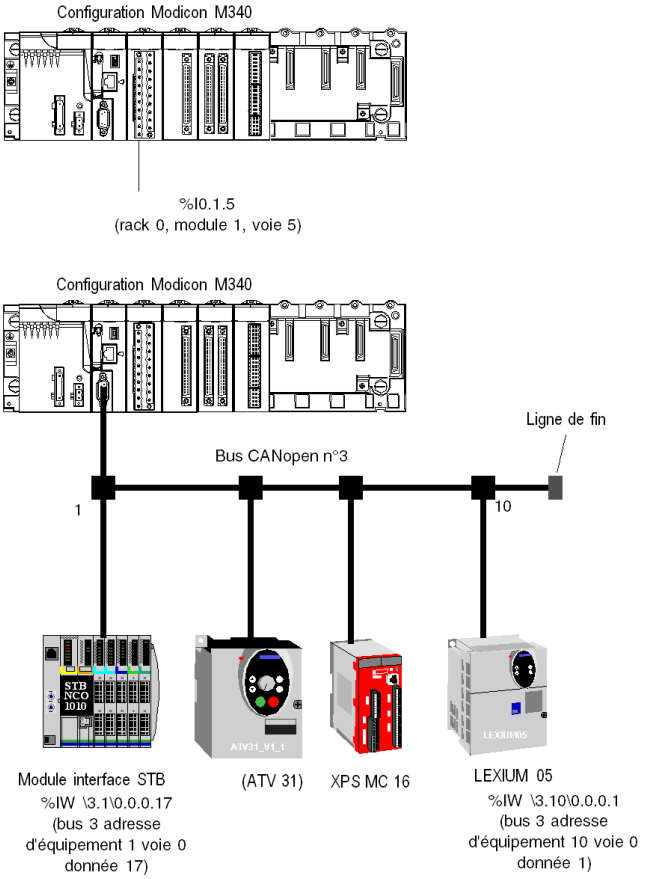
<r> numéro de rack.

<m> emplacement du module

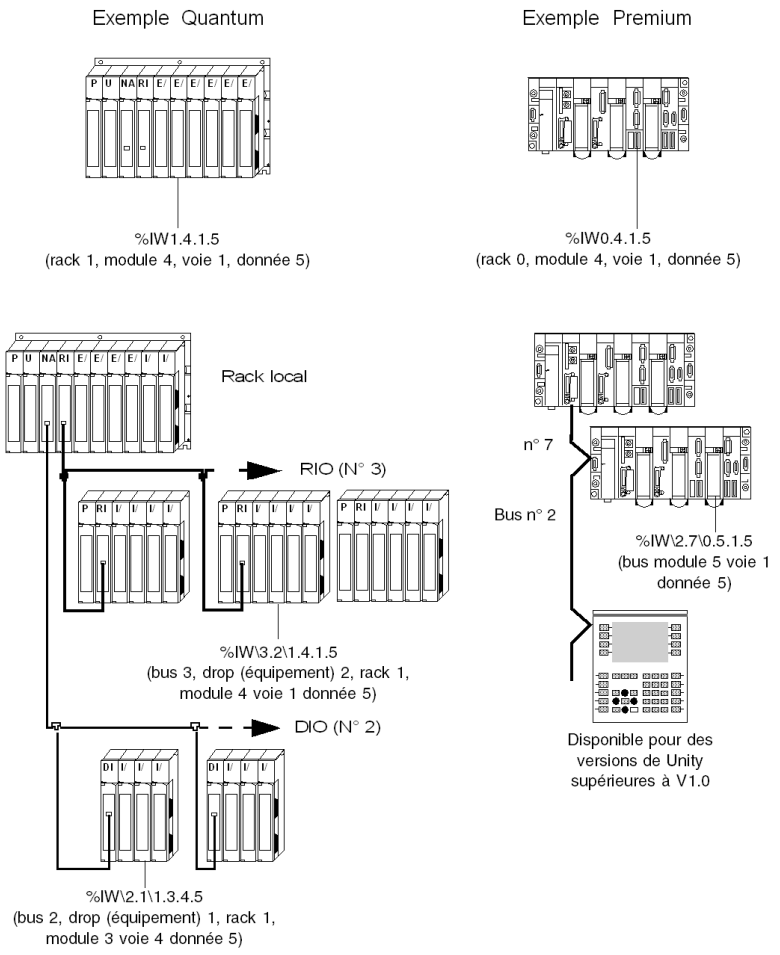
<c> numéro de voie (0 à 999) ou mot réservé MOD.

<d> numéro donnée (0 à 999) ou mot réservé ERR (facultatif si valeur 0). Pour Modicon M340 <d> est toujours pair.

Exemples : station locale et station sur bus pour automates Modicon M340



Exemples : station locale et station sur bus pour automates Quantum et Premium.



Références de données

10

Contenu du chapitre

Ce chapitre décrit les références d'instances de données.

Ces références peuvent être:

- des références par valeurs,
- des références par noms,
- des références par adresses.

Contenu de ce chapitre

Ce chapitre contient les sujets suivants :

Références des instances de données par valeur

Présentation

Qu'est ce qu'une référence d'instance de données? *(voir page 237)*

Présentation

Une référence d'instance de données par valeur est une instance ne possédant pas de nom (symbole) ou d'adresse topologique.

Il s'agit d'une **valeur immédiate** qui peut être affectée à une instance de type de données appartenant à la famille EDT.

La norme IEC 1131 autorise les valeurs immédiates sur des instances de données de type:

- Booléen
  - BOOL
  - EBOOL
- entiers
  - INT
  - UINT
  - DINT
  - UDINT
  - TIME
- réels
  - REAL
- dates et heures
  - DATE
  - DATE AND TIME
  - TIME_OF_DAY
- chaînes de caractères
  - STRING

Le logiciel de programmation étend la norme en y ajoutant les **types chaine de bits**.

- BYTE
- WORD
- DWORD

Exemples de valeurs immédiates :

Ce tableau associe des valeurs immédiates à des types d'instance

Valeur immédiate	**Type d'instance**
'Je suis une chaine de caractères'	STRING
T#1s	TIME
D#2000-01-01	DATE
TOD#12:25:23	TIME_OF_DAY
DT#2000-01-01-12:25:23	DATE_AND_TIME
16#FFF0	WORD
UINT#16#9AF **(valeur typée)**	UINT
DWORD#16#FFFF **(valeur typée)**	DWORD

Références des instances de données par nom

Présentation

Qu'est ce qu'une référence d'instance de données? *(voir page 237)*

Références d'instances de la famille EDT

L'utilisateur choisit un nom (symbole) qui permet d'accéder à l'instance des données :

```
Etat_Vanne: BOOL

Seuil_Haut: EBOOL AT %M10

Contenu_Tremie: UINT

Température_Four: INT AT %MW100

Valeur_Codeur: WORD
```

Références d'instances de la famille DDT

Cas des tableaux:

L'utilisateur choisit un nom (symbole) qui permet d'accéder à l'instance des données :

```
Soit 2 types de tableaux:

Game_Couleurs ARRAY[1..15]OF STRING
Véhicules ARRAY[1..100]OF Game_Couleurs
;

Voiture: Véhicules
Nom d'instance du tableau de type Véhicules

Voiture[11, 5]
Accés à la voiture 11 de couleur correspondant au 5 élément du
tableau Game_Couleurs
```

Cas des structures:

L'utilisateur choisit un nom (symbole) qui permet d'accéder à l'instance des données :

```
Soit les 2 structures:

ADRESSE
   Rue: STRING[20]
   Code_Postal: UDINT
   Ville: STRING: [20]

IDENT
   Nom: STRING[15]
   Prenom: STRING[15]
   Age: UINT
   Date_Naissance: DATE
   Localité: ADRESSE


Personne_1 :IDENT
;Nom d'instance de la stucture de type IDENT

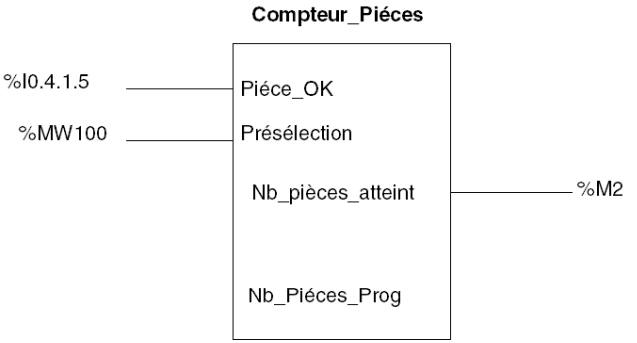
Personne_1.Age
Accés à l'age de Personne_1

Personne_1.Localité.Ville
;Accés à la localité où réside Personne_1
```

Références d'instances des familles DFB\EFB

L'utilisateur choisit un nom (symbole) qui permet d'accéder à l'instance des données :

Soit le type de DFB:



Compteur_Vis: Compteur_Pièces
Nom d'instance du bloc de type Compteur_Pièces

Compteur_Vis.Nb_Piéces_Prog
Accés à la variable publique Nb_Piéces_Prog

Compteur_Vis.Nb_piéces_atteint
Accés à l'interface de sortie Nb_piéces_atteint

Références d'instances de données par adresses

Introduction

Qu'est ce qu'une référence d'instance de données ? *(voir page 237)*

Objet de ce chapitre

Une référence d'instance de données par adresse n'est possible que sur certaines instances de données qui appartiennent à la **famille EDT**. Ces instances sont :

- les variables internes (%M<i>, %MW<i>, %MD<i>, %MF<i>),
- les constantes (%KW<i>, %KD<i>, %KF<i>),
- les entrées/sorties (%I<adresse>, %Q<adresse>).

NOTE : les instances %MD<i>, %MF<i>, %KD<i> et %KF<i> ne sont pas disponibles pour les automates Modicon M340.

Référence par adressage direct

L'adressage est considéré direct quand l'adresse de l'instance est fixe, c'est-à-dire définie lors de l'écriture dans le programme.

Exemples :

```
%M1
Accès au premier bit de la mémoire

%MW12
Accès au douzième mot de la mémoire

%MD4
Accès au quatrième double mot de la mémoire

%KF100
; Accès au centième mot flottant de la mémoire

%Q0.4.0.5
Accès au cinquième bit du module de sortie en position 4
du rack 0
```

Références par adresse indexée

L'adressage est considéré indexé lorsque l'adresse de l'instance est complétée par un index.

L'index est défini soit par :

- une valeur appartenant aux types Entiers,
- une expression arithmétique composée de types Entiers.

Une variable indexée a toujours une équivalence non indexée :

```
%MW<i > [ < index >] < = > %MW<j >
```

Les règles de calcul de <j> sont les suivantes :

Objet<i>[index]	**Objet<j>**
%M<i>[index]	<j>=<i> + <index>
%MW<i>[index]	<j>=<i> + <index>
%KW<i>[index]	<j>=<i> + <index>
%MD<i>[index]	<j>=<i> + (<index> x 2)
%KD<i>[index]	<j>=<i> + (<index> x 2)
%MF<i>[index]	<j>=<i> + (<index> x 2)
%KF<i>[index]	<j>=<i> + (<index> x 2)

Exemples :

```
%MD6[10] < = > %MD26
```

```
%MW10[My_Var+8] < = > %MW20 (avec My_Var=2)
```

Lors de la compilation du programme, un contrôle vérifie que :

- l'index n'est pas négatif,
- l'index ne dépasse pas l'espace mémoire alloué pour chacun des trois types.

Bits extraits de mots

Il est possible d'extraire l'un des 16 bits des mots simples (%MW, %SW; %KW, %IW, %QW).

L'adresse de l'instance est complétée par le rang du bit extrait (<j>).

```
WORD < i > . < j >
```

Exemples :

%MW10.4
Bit n°4 du mot %MW10

%SW8.4
Bit n°4 du mot système %SW8

%KW100.14
Bit n°14 de la constante KW100

%QW0.5.1.0.10
Bit n°10 du mot 0 de la voie 1 du module de sortie 5 du rack 0

Bits extraits d'octets

Il est possible d'extraire l'un des bits d'un octet.

L'adresse du premier bit extrait est accessible par :

- le nom de l'octet correspondant,
- le rang définissant sa position dans l'octet (nombre entre 0 et 7).

Exemple :

MyByte est une variable de type BYTE. MyByte.i est une valeur BOOL valide si 0 <= i <= 7

MyByte.0, MyByte.3 et MyByte.7 sont des valeurs BOOL valides.

MyByte.8 n'est pas valide.

Tableaux de bits et de mots

Il s'agit d'une suite d'objets adjacents (bits ou mots) de même type et de longueur définie.

```
OBJECT<i> : L
```

Présentation des tableaux de bits :

Type	Adresse	Accès en écriture
Bits d'entrées TOR	%Ix.i:L	Non
Bits de sorties TOR	%Qx.i:L	Oui
Bits internes	%Mi:L	Oui

Présentation des tableaux de mots :

Type	Adresse	Accès en écriture
Mots internes	%MWi:L %MDi:L %MFi:L	Oui
Mots constants	%KWi:L %KDi:L %KFi:L	Non
Mots système	%SW50:4	Oui

Exemples :

```
%M2 : 65
Définit un tableau de EBOOL démarrant à partir de %M2 jusqu'à %M66

%MW125 : 30
Définit un tableau de INT démarrant à partir de %MW125 jusqu'à %MW 154
```

Règles d'appellation des données

Introduction

Dans une application l'utilisateur choisit un nom pour:

- définir un type de données,
- instancier des données (symbole),
- identifier une section.

Certaines règles ont été définies afin d'empêcher les conflits. Pour cela, il faut différencier les différents domaines d'application des données

Qu'est ce qu'un domaine ?

Ils s'agit d'un espace de l'application à partir duquel une variables est accessible ou non, tel que:

- le domaine application qui comprend:
  - les différentes tâches de l'application,
  - les sections qui les composent.
- le domaines par types de données tels que:
  - les structures/tableaux pour la famille DDT,
  - les EFB\DFB pour la famille blocs fonctions

Régles

Ce tableau définit s'il est possible ou non d'utiliser un **nom** déjà existant dans l'application pour des éléments nouvellement créés:

Contenu de l'application --> Nouveaux éléments (ci-dessous)	**Section**	**SR**	**DDT/IO DDT**	**Type FB**	**Instances FB**	**EF**	**Variable**
Section	Non	Non	Oui	Oui	Oui	Oui	Oui
SR	Non	Non	Oui	Oui	Non	(1)	Non
DDT/IODDT	Non	Non	Non	Non (4)	Non	Non (4)	Non
Type FB	Oui	Oui	Non	Non	(3)	Non	(3)
Instances FB	Non	Non	Non	Oui	Non	Oui	Non
EF	Oui	(2)	Non	Non	Non	Non	Non
Variable	Oui	Non	Oui	Oui	Non	(1)	Non

(1) : Une instance appartenant au domaine application ne peut pas avoir le même nom qu'une EF.

(2) : Une instance appartenant au domaine du type (variable interne) peut avoir le même nom qu'une EF. L'EF en question ne peut pas être utilisée avec ce type.

(3) : La création ou l'importation d'EFB/DFB ayant le même nom qu'une instance existante sont interdites.

(4) : Un élément DDT/IODDT pourrait avoir le même nom qu'un FB/EF, mais ce n'est pas recommandé car le FB/EF ne pourrait pas être utilisé dans l'application.

NOTE : Ci-dessous un complément aux règles présentées dans le tableau, qui précisent que :

- à l'intérieur d'un type, une instance (variable interne) ne peut avoir le même nom que le nom de type de l'objet auquel elle appartient,
- il n'y a pas de conflit entre le nom d'une instance appartenant à une section de l'application et le nom d'une instance appartenant à une section d'un DFB,
- il n'y a pas de conflit entre le nom d'une section appartenant à une tâche et le nom d'une section appartenant à un DFB.

Langage de programmation

IV

Contenu de cette partie

Cette partie décrit la syntaxe des langages de programmation disponibles.

Contenu de cette partie

Cette partie contient les chapitres suivants :

Langage à blocs fonction FBD

11

Objet de ce sous-chapitre

Ce chapitre décrit le langage à blocs fonction FBD conforme à la norme IEC 61131.

Contenu de ce chapitre

Ce chapitre contient les sujets suivants :

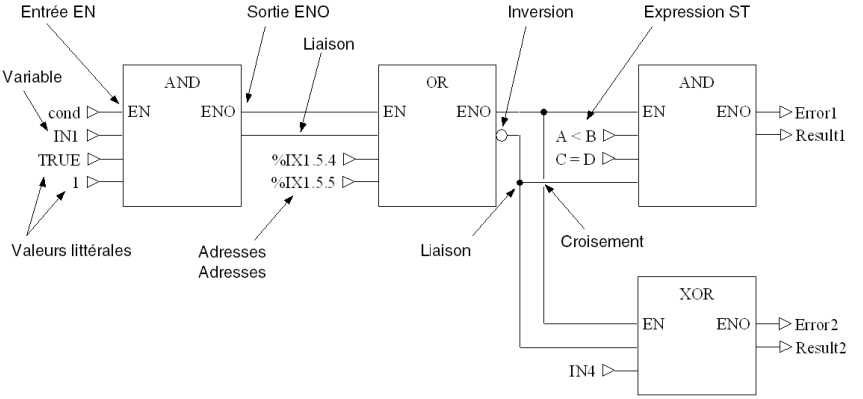
Informations générales sur le langage à blocs fonction FBD

Présentation

L'éditeur FBD permet la programmation graphique de blocs fonction conformément à la norme CEI 61131-3.

Représentation d'une section FBD

Représentation :



Objets

Les objets du langage FBD (diagramme de blocs fonctionnels) offrent des aides permettant de structurer une section en un ensemble de :

- EF et EFB (fonctions élémentaires *(voir page 330)* et blocs fonction élémentaires *(voir page 331)*),
- DFB (blocs fonction dérivés) *(voir page 332)*,
- procédures *(voir page 332)* et
- contrôles *(voir page 342)*.

Ces objets, regroupés sous l'abréviation générique FFB, peuvent être liés les uns aux autres par :

- des liaisons *(voir page 344)* ou
- des paramètres réels *(voir page 333)*.

La logique de la section peut être commentée par des objets texte (voir *Objet texte, page 346*).

Taille de la section

Une section FBD comprend une fenêtre incluant une seule page.

Cette page est placée sur une grille. Une unité de grille comprend 10 points de trame. Une unité de trame est l'espace le plus petit possible entre deux objets d'une section FBD.

Le langage FBD n'est pas basé sur les cellules les objets sont toutefois ajustés sur les points de trame.

Une section FBD peut être configurée en nombre de cellules (points de trame horizontaux et points de trame verticaux).

Conformité CEI

Pour la description de la conformité CEI du langage FBD, voir Conformité CEI *(voir page 657)*.

Fonctions élémentaires, blocs fonction élémentaires, blocs fonction dérivés et procédures (FFB)

Introduction

FFB est le terme générique pour :
- les fonctions élémentaires (EF) *(voir page 330)*
- les blocs fonction élémentaires (EFB) *(voir page 331)*
- les DFB (blocs fonction dérivés) *(voir page 332)*
- Procédure *(voir page 332)*

Fonction élémentaire

Les fonctions élémentaires (EF) n'ont pas d'état interne. Lorsque les valeurs d'entrée sont identiques, la valeur de sortie est la même à chaque exécution de la fonction. Par exemple, l'addition de deux valeurs donne toujours le même résultat.

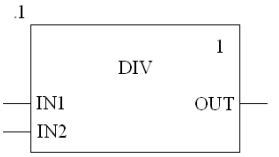
Une fonction élémentaire est représentée graphiquement comme un cadre avec des entrées et une sortie. Les entrées sont toujours représentées sur la gauche et la sortie toujours sur la droite du cadre.

Le nom de la fonction, c'est-à-dire le type de fonction, est affiché au centre du cadre.

Le numéro d'exécution *(voir page 347)* de la fonction apparaît à droite du type de fonction.

Le numéro de fonction est affiché au-dessus du cadre. Le numéro de fonction représente le numéro courant de la fonction dans la section actuelle. Les numéros de fonction ne peuvent pas être modifiés.

Fonction élémentaire



Pour certaines fonctions élémentaires, il est possible d'augmenter le nombre d'entrées.

Bloc fonction élémentaire

Les blocs fonction élémentaires (EFB) ont des états internes. Lorsque les valeurs d'entrée sont identiques, la valeur de sortie peut être différente pour toutes les exécutions de la fonction. Par exemple, pour un compteur, la valeur de sortie augmente.

Un bloc fonction élémentaire est représenté graphiquement sous forme de cadre avec des entrées et des sorties. Les entrées sont toujours représentées sur la gauche et les sorties toujours sur la droite du cadre.

Les blocs fonction peuvent avoir plusieurs sorties.

Le nom du bloc fonction, c'est-à-dire le type de bloc fonction, est affiché au centre du cadre.

Le numéro d'exécution *(voir page 347)* du bloc fonction apparaît à droite du type de bloc fonction.

Le nom d'instance est affiché au-dessus du cadre.

Le nom d'instance permet d'identifier précisément le bloc fonction dans un projet.

Le nom d'instance est généré automatiquement et présente la structure suivante : `FBI_n`

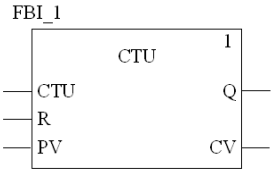
`FBI` = Instance de bloc fonction

`n` = numéro courant du bloc fonction au sein du projet

Vous pouvez modifier ces noms générés automatiquement pour rendre la vue d'ensemble plus claire. Le nom d'instance (32 caractères maximum) doit être unique dans tout le projet ; aucune distinction n'est faite ici entre majuscules et minuscules. Le nom d'instance doit respecter les conventions de noms générales.

NOTE : conformément à la norme CEI 61131-3, les noms d'instance doivent commencer par une lettre. Si vous voulez également utiliser des chiffres comme premier caractère, vous devez activer cette fonction.

Bloc fonction élémentaire

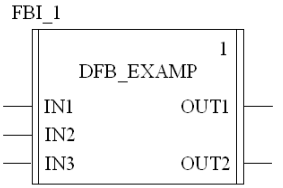


DFB

Les blocs fonction dérivés (DFB) ont les mêmes caractéristiques que les blocs fonction élémentaires. Ils sont cependant créés par l'utilisateur dans les langages FBD, LD, IL et/ou ST.

L'unique différence par rapport aux blocs fonction élémentaires est que le bloc fonction dérivé est représenté graphiquement sous forme de cadre avec deux lignes verticales.

Bloc fonction dérivé



Procédure

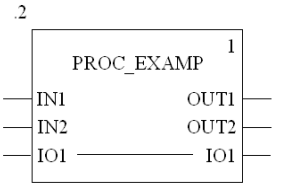
Techniquement, les procédures sont des fonctions.

L'unique différence par rapport aux fonctions élémentaires est que les procédures peuvent comprendre plus d'une sortie et qu'elles prennent en charge le type de données `VAR_IN_OUT`.

Les procédures sont une extension de la norme CEI 61131-3 et doivent être activées de manière explicite.

Il n'y a pas de différence physique entre les procédures et les fonctions élémentaires.

Procédure

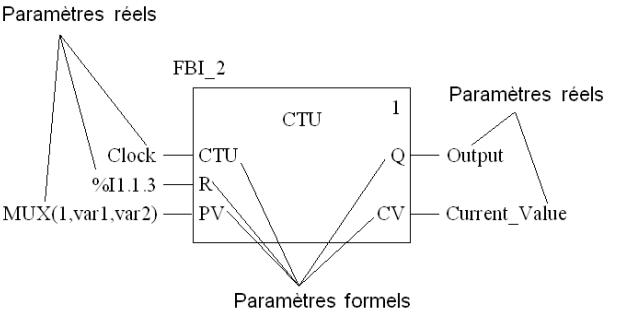


Paramètres

Pour importer des valeurs dans le FFB ou exporter des valeurs du FFB, des entrées et des sorties sont nécessaires. Elles sont appelées paramètres formels.

Les paramètres formels sont liés à des objets qui comprennent les états courants du traitement. Ces états sont appelés paramètres réels.

Paramètres formels et réels :



Durant l'exécution du programme, les valeurs sont transmises, par le biais des paramètres réels, du processus au FFB, et renvoyées à nouveau à la sortie après le traitement.

Seul un objet (paramètre réel) du type de données suivant peut être relié aux entrées FFB :

- variable
- adresse
- libellé
- expression ST *(voir page 509)*
  Les expressions ST des entrées FFB représentent une extension de la norme CEI 61131-3 et doivent être activées de manière explicite.
- Liaison

Les combinaisons d'objets (paramètres réels) suivantes peuvent être reliées aux sorties FFB :

- une variable
- une variable et une ou plusieurs liaisons (non valable pour les sorties VAR_IN_OUT *(voir page 339)*)
- une adresse,
- une adresse et une ou plusieurs liaisons (non valable pour les sorties VAR_IN_OUT *(voir page 339)*)
- une ou plusieurs liaisons (non valable pour les sorties VAR_IN_OUT *(voir page 339)*)

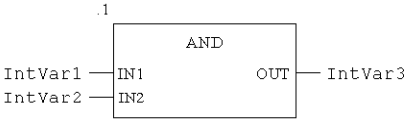
Le type des données de l'objet à relier doit correspondre au type des données de l'entrée/la sortie FFB. On choisira un type de données adapté pour le bloc fonction, si tous les paramètres réels sont constitués de valeurs littérales.

Exception : pour les entrées/sorties génériques FFB de type de données ANY_BIT, des objets de type de données INT ou DINT (pas UINT ni UDINT) peuvent être reliés.

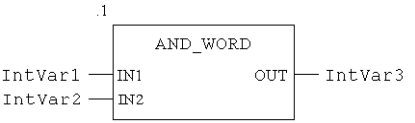
Il s'agit d'une extension de la norme CEI 61131-3 et doit donc être activée de manière explicite.

Exemple :

Autorisé :



Non autorisé :



(Dans ce cas AND_INT doit être utilisé.)

Il n'est en principe pas nécessaire d'affecter un paramètre réel à tous les paramètres formels. Cependant, cela n'est pas valable pour les broches inversées. Un paramètre réel doit toujours leur être affecté. Cela est également impératif pour certains types de paramètre formel. Pour connaître les types concernés, veuillez vous reporter au tableau suivant.

Tableau des types de paramètre formel :

Type de paramètre	EDT	STRING	ARRAY	ANY_ARRAY	IODDT	STRUCT	FB	ANY
EFB : Entrée	-	+	+	+	/	+	/	+
EFB : VAR_IN_OUT	+	+	+	+	+	+	/	+
EFB : Sortie	-	-	+	+	+	-	/	+
DFB : Entrée	-	+	+	+	/	+	/	+
DFB : VAR_IN_OUT	+	+	+	+	+	+	/	+
DFB : Sortie	-	-	+	/	/	-	/	+
EF : Entrée	-	-	+	+	+	+	+	+
EF : VAR_IN_OUT	+	+	+	+	+	+	/	+
EF : Sortie	-	-	-	-	-	-	/	-
Procédure : Entrée	-	-	+	+	+	+	+	+

Type de paramètre	EDT	STRING	ARRAY	ANY_ARRAY	IODDT	STRUCT	FB	ANY
Procédure : VAR_IN_OUT	+	+	+	+	+	+	/	+
Procédure : Sortie	-	-	-	-	-	-	/	+
+ paramètre réel impératif								
- Paramètre réel non impératif								
/ Non applicable								

Les FFB utilisant aux entrées des paramètres réels, auxquels aucune valeur n'a encore été affectée, fonctionnent avec les valeurs initiales de ces paramètres réels.

Si aucune valeur n'est affectée à un paramètre formel, la valeur initiale est utilisée pendant l'exécution du bloc fonction. Si aucune valeur initiale n'est définie, la valeur par défaut (0) est utilisée.

Si aucune valeur n'est affectée à un paramètre formel et que le bloc fonction/DFB a été instancié à plusieurs reprises, les instances appelées par la suite travaillent avec l'ancienne valeur.

Variables publiques

Certains blocs fonction disposent non seulement d'entrées et de sorties, mais également de variables publiques.

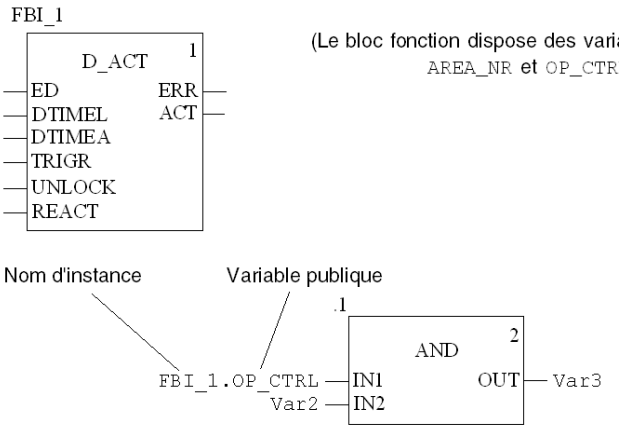
Ces variables permettent de transmettre des valeurs statiques (valeurs non influencées par le procédé) au bloc fonction. Elles sont donc utilisées lors du paramétrage du bloc fonction.

Les variables publiques sont une extension de la norme CEI 61131-3.

Les valeurs sont affectées aux variables publiques via leur valeur initiale.

Les valeurs des variables publiques sont ensuite lues à partir du nom d'instance du bloc fonction et du nom de la variable publique.

Exemple :



Variables privées

Certains blocs fonction disposent non seulement d'entrées, de sorties et de variables publiques, mais également de variables privées.

A l'instar des variables publiques, ces variables permettent de transmettre des valeurs statistiques (valeurs non influencées par le procédé) au bloc fonction.

Le programme utilisateur ne peut pas accéder à ces variables. Seule la table d'animation en a la capacité.

NOTE : les DFB imbriqués sont déclarés comme des variables privées du DFB parent. Ainsi, leurs variables ne sont pas accessibles via la programmation, mais via la table d'animation.

Les variables privées sont une extension de la norme CEI 61131-3.

Remarques sur la programmation

Veuillez tenir compte des remarques qui suivent sur la programmation :

- Les FFB ne sont exécutées que lorsque l'entrée EN = 1 ou lorsque l'entrée `EN` est désactivée (voir aussi EN et ENO *(voir page 337)*).
- Les entrées et sorties booléennes peuvent être inversées.
- Des conditions particulières s'appliquent lors de l'utilisation de variables VAR_IN_OUT *(voir page 339)*.
- Les instances de DFB ou bloc fonction peuvent être appelées à plusieurs reprises (voir aussi Appel multiple d'une instance de bloc fonction *(voir page 337)*).

Appel multiple d'une instance de bloc fonction

Les instances de DFB ou bloc fonction peuvent être appelées à plusieurs reprises, à l'exception des instances d'EFB de communication et blocs fonction/DFB ayant une sortie ANY mais pas d'entrée ANY, qui ne peuvent être appelées qu'une seule fois.

L'appel multiple de la même instance de DFB/bloc fonction est par exemple utile dans les cas suivants :

- si le bloc fonction/DFB ne comporte aucune valeur interne ou si celle-ci n'est plus nécessaire pour un traitement ultérieur.
  Dans ce cas, l'appel multiple de la même instance de DFB/bloc fonction permet d'économiser de l'espace mémoire, car le code du bloc fonction/DFB n'est alors chargé qu'une seule fois.
  Le bloc fonction/DFB est pour ainsi dire traité comme une "fonction".
- si le bloc fonction/DFB comprend des valeurs internes et que celles-ci doivent être influencées à différents endroits du programme, la valeur d'un compteur, par exemple, doit être augmentée à différents endroits du programme.
  Dans ce cas, l'appel multiple de la même instance de bloc fonction/DFB permet d'économiser la mémoire des résultats intermédiaires pour un traitement ultérieur à un autre endroit du programme.

EN et ENO

Une entrée EN et une sortie ENO peuvent être configurées pour tous les FFB.

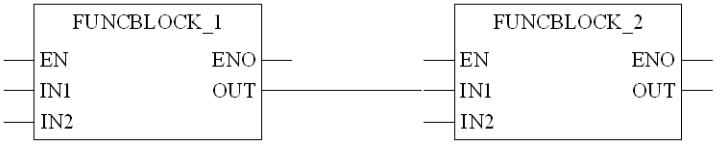
Si la valeur de EN est déjà réglé sur 0, lors de l'appel de FFB, les algorithmes définis par FFB ne sont pas exécutés et ENO est réglé sur 0.

Si la valeur de EN est déjà réglé sur 1, lors de l'appel de FFB, les algorithmes définis par FFB sont exécutés. Après l'exécution sans erreur de ces algorithmes, la valeur de ENO est réglée sur 1. Si une erreur se produit durant l'exécution de ces algorithmes, ENO est réglé sur 0.

Si aucune valeur n'est attribuée à la broche EN à l'appel du FFB, l'algorithme défini par ce dernier est exécuté (comme lorsque EN a la valeur « 1 »). Reportez-vous à la section *Maintenir les liens de sortie sur les EF désactivés (voir Unity Pro, Modes de marche,)*.

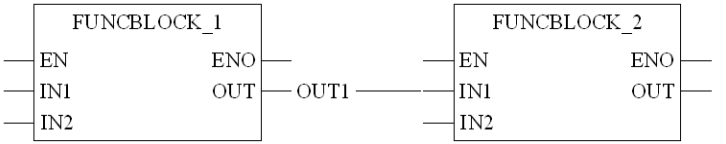
Si ENO est réglé sur 0 (car EN = 0 ou en raison d'une erreur d'exécution) :

- Blocs fonction
  - Traitement EN/ENO pour les blocs fonction ayant (seulement) une liaison comme paramètre de sortie :



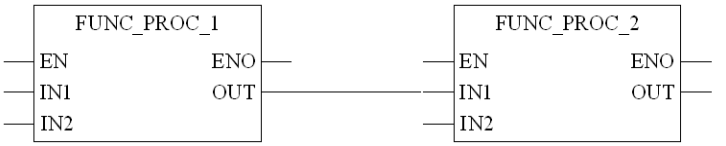
Lorsque EN est réglé sur 0 par FUNCBLOCK_1, la liaison à la sortie OUT de FUNCBLOCK_1 conserve l'ancien statut qu'elle avait lors du dernier cycle correct.

- Traitement EN/ENO pour les blocs fonction ayant une variable et une liaison comme paramètre de sortie :



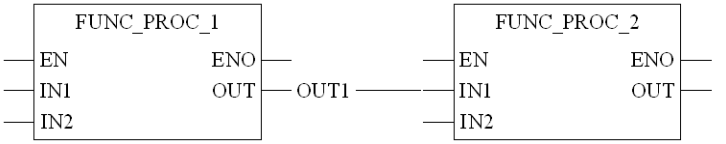
Lorsque EN est réglé sur 0 par FUNCBLOCK_1, la liaison à la sortie OUT de FUNCBLOCK_1 conserve l'ancien statut qu'elle avait lors du dernier cycle correct. La variable OUT1 située sur la même broche conserve son ancien statut ou peut être modifiée depuis l'extérieur sans avoir d'influence sur la liaison. La variable et la liaison sont enregistrées indépendamment l'une de l'autre.

- Fonctions/Procédures
  Selon la définition CEI 61131-3, les sorties de fonctions désactivées (entrée EN mise à "0") sont indéfinies. (Le même principe s'applique aux procédures.)
  Voici, néanmoins, une explication des états des sorties dans un tel cas :
  - Traitement EN/ENO pour les fonctions/procédures ayant (seulement) une liaison comme paramètre de sortie :



Si EN de FUNC_PROC_1 est réglé sur 0, la valeur de la liaison sur la sortie OUT de FUNC_PROC_1 dépend du paramètre de projet **Maintenir les liens de sortie sur les EF désactivés**, disponible depuis Unity Pro 4.1.
Si ce paramètre de projet est réglé sur 0, la valeur de la liaison est réglée sur 0.
Si ce paramètre de projet est réglé sur 1, la liaison conserve la valeur qu'elle avait lors du dernier cycle exécuté correctement.
Consultez la section *Maintenir les liens de sortie sur les EF désactivés (voir Unity Pro, Modes de marche,)*.

  - Traitement EN/ENO pour les fonctions/procédures ayant une variable et une liaison comme paramètre de sortie :



Si EN de FUNC_PROC_1 est réglé sur 0, la valeur de la liaison sur la sortie OUT de FUNC_PROC_1 dépend du paramètre de projet **Maintenir les liens de sortie sur les EF désactivés**, disponible depuis Unity Pro 4.1.
Si ce paramètre de projet est réglé sur 0, la valeur de la liaison est réglée sur 0.
Si ce paramètre de projet est réglé sur 1, la liaison conserve la valeur qu'elle avait lors du dernier cycle exécuté correctement.
Consultez la section *Maintenir les liens de sortie sur les EF désactivés (voir Unity Pro, Modes de marche,)*.
La variable OUT1 située sur la même broche conserve son ancien statut ou peut être modifiée depuis l'extérieur sans avoir d'influence sur la liaison. La variable et la liaison sont enregistrées indépendamment l'une de l'autre.

Le comportement aux sorties des FFB est indépendant du fait que les FFB soient appelés sans EN/ENO ou avec EN = 1.

NOTE : pour les blocs fonction désactivés (EN = 0) équipés d'une fonction d'horloge interne (par exemple, le bloc fonction DELAY), le temps semble continuer de s'écouler, car il est calculé à l'aide d'une horloge système et est, par conséquent, indépendant du cycle du programme et de la libération du bloc.

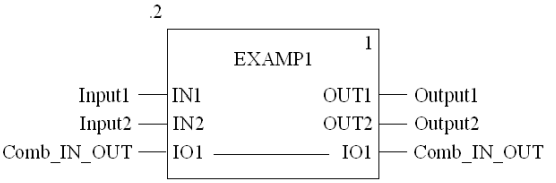
Variable VAR_IN_OUT

Les FFB sont souvent utilisés pour lire une variable à l'entrée (variables d'entrée), pour la traiter et pour transmettre de nouveau les valeurs modifiées de cette **même** variable (variables de sortie).

Ce cas exceptionnel d'une variable d'entrée/de sortie est également appelé variable VAR_IN_OUT.

Dans le FFB, une ligne indique que les variables d'entrée et de sortie sont liées l'une à l'autre.

Variable VAR_IN_OUT



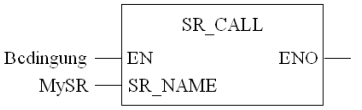
Il convient de noter les particularités suivantes en cas d'utilisation de FFB avec des variables VAR_IN_OUT :

- une variable doit être affectée à toutes les entrées VAR_IN_OUT.
- les liaisons graphiques permettent uniquement de relier des sorties VAR_IN_OUT à des entrées VAR_IN_OUT.
- seule une liaison graphique unique peut être reliée à une entrée/sortie VAR_IN_OUT.
- une combinaison de variables/d'adresses et de liaisons graphiques n'est pas possible pour les sorties VAR_IN_OUT.
- il est interdit de relier des valeurs littérales ou des constantes à des entrées/sorties VAR_IN_OUT.
- il est interdit d'utiliser des négations au niveau des entrées/sorties VAR_IN_OUT.
- des variables/composantes de variables différentes peuvent être reliées à l'entrée VAR_IN_OUT et à la sortie VAR_IN_OUT. Dans un tel cas, la valeur de la variable/composante de variable à l'entrée est copiée dans la variable/composante de variable à la sortie.

Appels de sous-programme

Appel d'un sous-programme

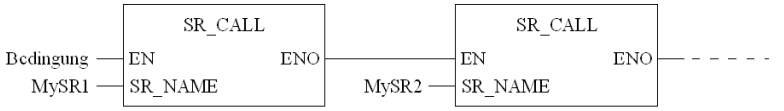
En FBD, les sous-programmes sont appelés à l'aide du bloc ci-dessous.



Si 1 est l'état de `EN`, le sous-programme correspondant (nom des variables à `SR_Name`) est appelé.

Pour ce type de bloc, la sortie `ENO` ne sert pas à afficher l'état d'erreur. Pour ce type de bloc, la sortie `ENO` est toujours 1 et permet d'appeler simultanément plusieurs sous-programmes.

La construction suivante permet d'appeler simultanément plusieurs sous-programmes.



Le sous-programme à appeler doit se trouver dans la même tâche que la section FBD appelante.

Il est possible d'appeler des sous-programmes au sein de sous-programmes.

Les appels de sous-programmes sont un complément de la norme CEI 61131-3 et doivent être activés de manière explicite.

Dans les sections d'actions SFC, les appels de sous-programmes ne sont autorisés que si le mode Multitoken a été activé.

Contrôles

Introduction

Les éléments de commande servent à l'exécution de sauts au sein d'une section FBD et au retour prématuré dans le programme principal depuis un sous-programme (SRx) ou un bloc fonction dérivé (DFB).

Contrôles

Les contrôles suivants sont disponibles.

Désignation	Représentation	Description
Jump		Si l'état de la liaison gauche est 1, un saut est exécuté jusqu'à l'étiquette (dans la section courante). Pour générer un saut conditionnel, l'objet saut est lié à une sortie FFB booléenne. Pour générer un saut inconditionnel, la valeur 1 est affectée à l'objet saut via la fonction `AND`.
Libellé	`LABEL:`	Les repères (destinations de saut) sont représentés comme du texte avec deux-points à la fin. Le texte est limité à 32 caractères et doit être unique dans l'ensemble de la section. Le texte doit respecter les conventions de nommage générales. Les étiquettes de saut ne peuvent être placées qu'entre les deux premiers points de trame sur la marge gauche de la section. **Note :** Les étiquettes de saut ne doivent "couper" aucun réseau, c'est-à-dire qu'une ligne imaginaire entre l'étiquette de saut et la marge droite de la section ne doit être coupée par aucun objet. Cela est également valable pour les liaisons.

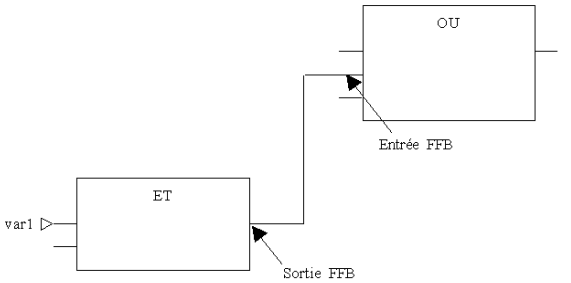
Désignation	Représentation	Description

Return		Des objets RETURN ne peuvent pas être utilisés dans le programme principal.
		● Dans un DFB, un objet RETURN force le retour au programme qui a appelé le DFB.
		  ● Le reste de la section de DFB contenant l'objet RETURN n'est pas exécuté.
		  ● Les sections suivantes du DFB ne sont pas exécutées.
		  Le programme qui a appelé le DFB sera exécuté après retour du DFB.
		  Si le DFB est appelé par un autre DFB, le DFB appelant sera exécuté après le retour.
		● Dans un SR, un objet RETURN force le retour au programme qui a appelé le SR.
		  ● Le reste du SR contenant l'objet RETURN n'est pas exécuté.
		  Le programme qui a appelé le SR sera exécuté après retour du SR.

Liaison

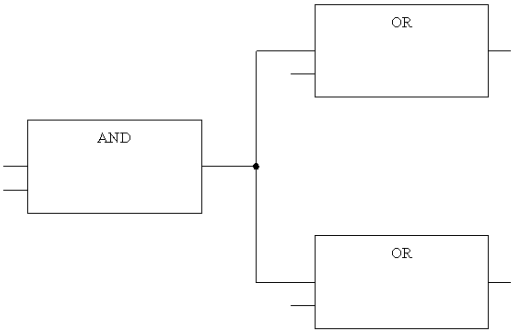
Description

Les liaisons sont des connexions verticales et horizontales entre les FFB.

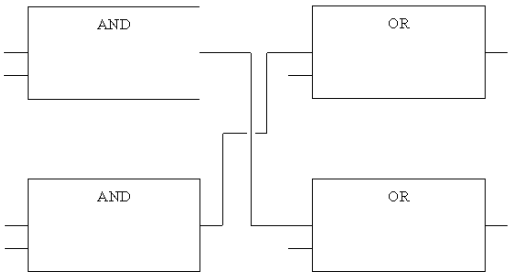


Représentation

Les points de liaison sont marqués par un cercle rempli.



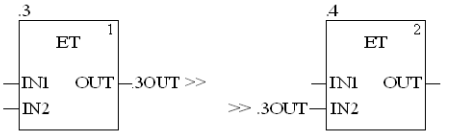
Les croisements sont représentés par une liaison interrompue.



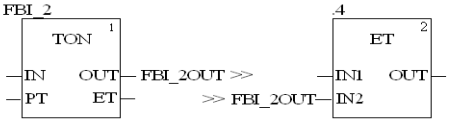
Remarques sur la programmation

Veuillez observer les remarques qui suivent sur la programmation :

- Les liaisons peuvent être utilisées pour chaque type de données.
- Les types de données respectifs des entrées/sorties à relier doivent correspondre les uns aux autres.
- Plusieurs liaisons peuvent être reliées à une sortie FFB, une seule cependant avec une entrée FFB.
- Seules des entrées et sorties peuvent être reliées ensemble. La liaison de plusieurs sorties n'est pas possible. Cela signifie qu'aucun lien OU via des liaisons n'est possible dans FBD. Il faut toujours utiliser une fonction `OU`.
- Le chevauchement des liaisons avec d'autres objets est admis.
- les boucles ne peuvent pas être configurées par le biais de liaisons, étant donné que, dans ce cas, l'ordre d'exécution dans la section ne peut pas être défini de façon unique. Les boucles doivent être résolues par le biais de paramètres réels (voir *Configuration de boucles, page 353*).
- Afin d'éviter le croisement de liaisons, ces dernières peuvent également être représentées sous la forme de connecteurs.
  A cette occasion, la source et la cible de la liaison sont caractérisées par un nom unique au sein de la section.
  Suivant le type d'objet source, le nom du connecteur est formé comme suit :
  - pour les fonctions : "numéro de fonction/paramètre formel" de la source de la liaison.



  - pour les blocs fonction : "nom d'instance/paramètre formel" de la source de la liaison.



Objet texte

Description

Dans le langage blocs fonctions FBD, les textes peuvent être placés sous forme d'objets texte. La taille de ces objets texte est fonction de la longueur du texte. Selon la longueur du texte, la taille de l'objet peut être agrandie, dans les sens vertical et horizontal, d'unités de grille supplémentaires. Les objets texte ne doivent pas se chevaucher avec des FFB, le chevauchement avec des liaisons est toutefois admis.

Ordre d'exécution des FFB

Introduction

L'ordre d'exécution est défini par la position des FFB dans la section (exécution de gauche à droite et de haut en bas). Lorsque, par la suite, les FFB sont liés à des liaisons graphiques, l'ordre d'exécution est alors déterminé par le flux de signaux.

Le numéro d'exécution (numéro figurant dans le coin supérieur droit du cadre de FFB) indique l'ordre d'exécution.

Ordre d'exécution des réseaux

Les règles suivantes s'appliquent à l'ordre d'exécution des réseaux :

- l'exécution d'une section a lieu réseau pour réseau via la liaison des FFB du haut vers le bas.
- les boucles ne peuvent pas être configurées par le biais de liaisons, étant donné que, dans ce cas, l'ordre d'exécution ne peut pas être défini de façon unique. Les boucles doivent être résolues par le biais de paramètres réels (voir *Configuration de boucles, page 353*).
- L'ordre d'exécution des réseaux qui ne sont pas reliés entre eux par des liaisons est défini par l'ordre graphique (de la partie supérieure droite vers la partie inférieure gauche). Vous pouvez influencer l'ordre d'exécution (voir *Modification de l'ordre d'exécution, page 349*).
- Le calcul d'un réseau doit être terminé entièrement avant que ne commence le calcul d'un autre réseau qui utilise les sorties du réseau précédent.
- Aucun élément d'un réseau n'est considéré comme calculé avant que l'état de toutes les entrées de cet élément n'ait été calculé.
- Le calcul d'un réseau est considéré comme terminé lorsque toutes les sorties de ce réseau sont calculées.

Flux de signaux dans un réseau

Les règles suivantes s'appliquent à l'ordre d'exécution au sein d'un réseau :

- Un FFB n'est calculé que lors tous les éléments (sorties FFB, etc) qui sont reliés à ses entrées sont calculés.
- L'ordre d'exécution des FFB reliés à différentes sorties du même FFB va du haut vers le bas.
- L'ordre d'exécution des FFB n'est pas influencé par leur position au sein du réseau.
  Cela ne s'applique pas lorsque plusieurs FFB sont reliés à la même sortie du FFB "à appeler". Dans ce cas l'ordre d'exécution est défini par l'ordre graphique (du haut vers le bas).

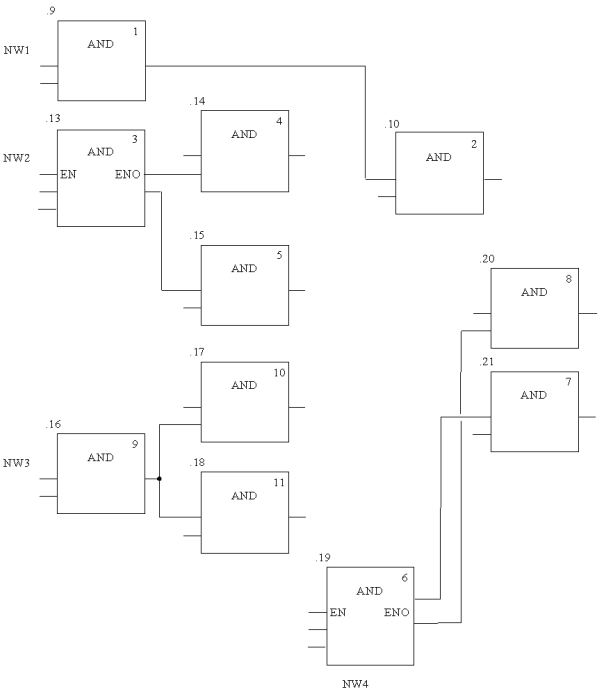
Priorités

Priorités lors de la détermination du flux de signaux au sein d'une section.

Priorité	Règle	Description
1	Lien	Les liaisons ont la priorité la plus élevée lors de la détermination du flux de signaux au sein d'une section FBD.
2	Définition utilisateur	Intervention de l'utilisateur sur l'ordre d'exécution.
3	Réseau par réseau	Le calcul d'un réseau doit être complètement terminé avant que le calcul du réseau suivant puisse commencer.
4	Ordre des sorties	Les FFB qui sont reliés aux sorties du même FFB "à appeler" sont calculés du haut vers le bas.
5	Rung par rung	Priorité la moins élevée. (Cela ne s'applique que si aucune autre règle n'intervient.)

Exemple

Exemple d'ordre d'exécution des objets dans une section FBD :



Modification de l'ordre d'exécution

Introduction

L'ordre d'exécution des réseaux et l'ordre d'exécution des objets au sein d'un réseau sont définis par une série de règles *(voir page 348)*.

Dans certains cas, il est nécessaire de modifier l'ordre d'exécution proposé par le système.

Pour définir/modifier l'ordre d'exécution des réseaux, vous disposez des possibilités suivantes :

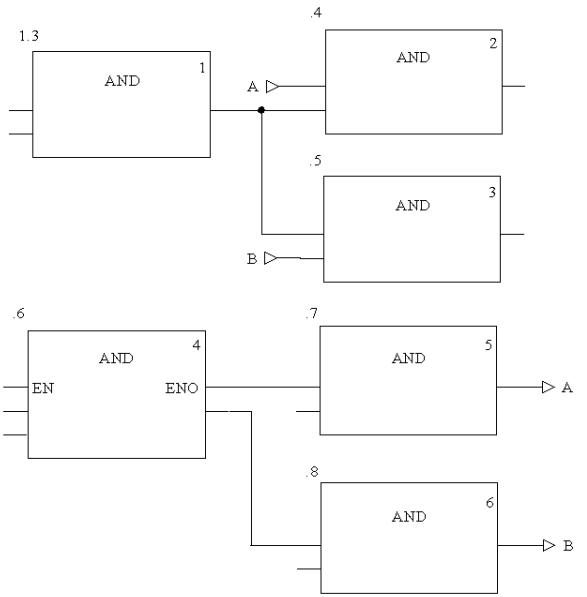
- utilisation de liaisons au lieu des paramètres réels,
- position des réseaux,
- détermination explicite de l'ordre d'exécution.

Pour définir/modifier l'ordre d'exécution des réseaux, vous disposez des possibilités suivantes :

- position des FFB.

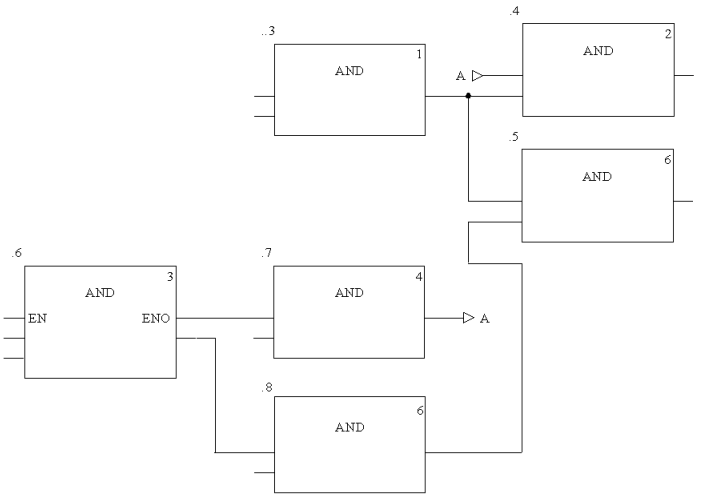
Situation initiale

Le schéma suivant représente deux réseaux dont l'ordre d'exécution est déterminé uniquement par leur position au sein de la section, même si les blocs `.4/.5` et `.7/.8` nécessitent un ordre d'exécution différent.



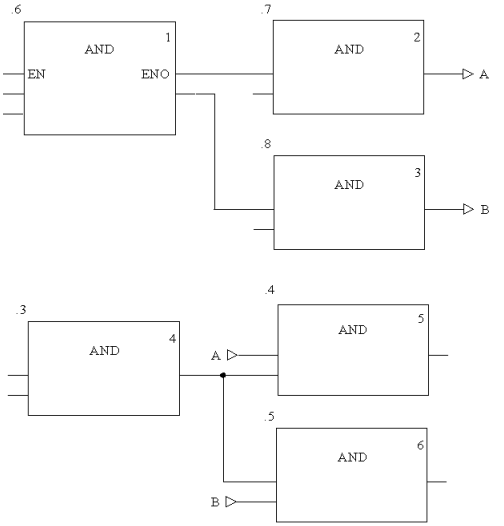
Liaison au lieu des paramètres réels

Lorsque l'on utilise une liaison au lieu d'une variable, les deux réseaux sont exécutés dans l'ordre correct (voir également *Situation initiale, page 349*).



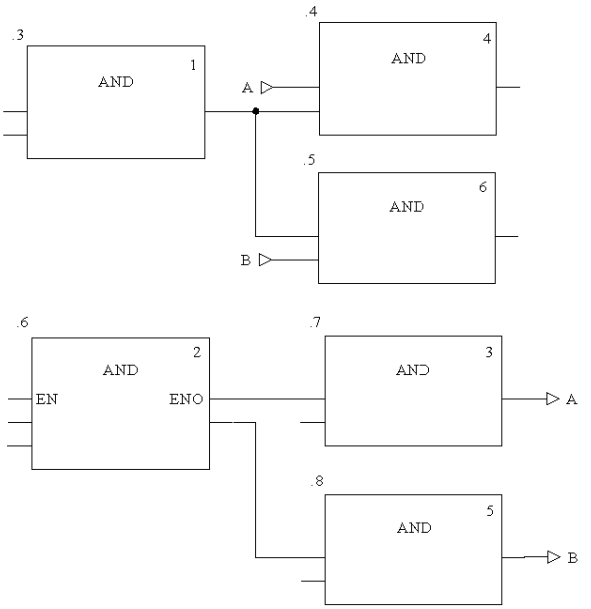
Position des réseaux

Vous pouvez obtenir l'ordre d'exécution correct en modifiant les positions des réseaux dans la section (voir également *Situation initiale, page 349*).



Détermination explicite

Vous pouvez obtenir l'ordre d'exécution correct en modifiant de manière explicite l'ordre d'exécution d'un FFB. Pour les FFB dont l'ordre d'exécution a été modifié de manière explicite, le numéro d'exécution s'affiche dans un champ noir (voir aussi *Situation initiale, page 349*).



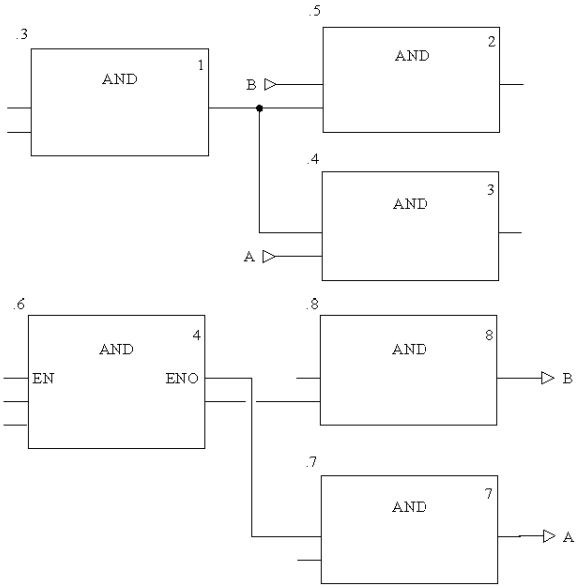
NOTE : Le système n'autorise qu'une seule référence par instance. Ainsi, l'instance ".7"par exemple ne peut être référencée qu'une fois.

Positions des FFB

La position des FFB n'a alors une influence sur l'ordre d'exécution que si plusieurs FFB sont reliés à la même sortie du FFB "à appeler" (voir également *Situation initiale, page 349*).

Les positions des blocs `.4` et `.5` sont permutées dans le premier réseau. Dans ce cas (source commune des deux entrées de bloc), l'ordre d'exécution des deux blocs est également permuté (traitement du haut vers le bas).

Les positions des blocs `.7` et `.8` sont permutées dans le deuxième réseau. Dans ce cas (source différente des entrées de bloc) l'ordre d'exécution des deux blocs n'est pas permuté (traitement dans l'ordre des sorties de bloc à appeler).

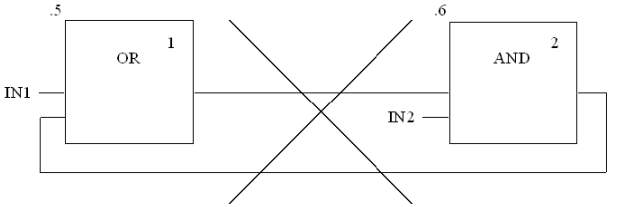


Configuration de boucles

Boucles non permises

La configuration de boucles exclusivement par le biais de liaisons n'est pas permise, étant donné que, dans ce cas, une détermination unique du flux de signaux n'est pas possible (la sortie d'un FFB est l'entrée du FFB suivant, et la sortie de celui-ci est à son tour l'entrée du premier).

Boucles non permises par le biais de liaisons



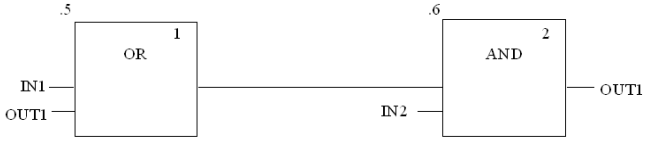
Résolution par le biais d'un paramètre réel

Une telle logique doit être résolue par le biais de variables de réaction, afin que le flux de signaux puisse être défini de façon unique.

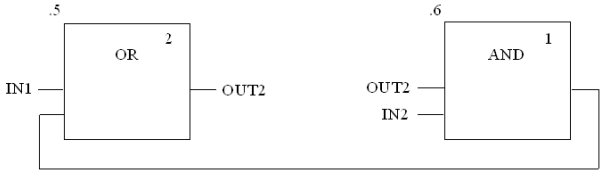
Les variables de réaction doivent être initialisées. La valeur initiale est utilisée lors de la première exécution de la logique. Une fois la première exécution effectuée, la valeur initiale est remplacée par la valeur actuelle.

Respectez pour les deux variantes l'ordre d'exécution (numéro entre parenthèses après le nom d'instance) des deux blocs.

Boucle résolue par le biais d'un paramètre réel : Variante 1



Boucle résolue par le biais d'un paramètre réel : Variante 2



Langage à contacts (LD)

12

Objet de ce sous-chapitre

Ce chapitre décrit le langage à contacts LD conforme à la norme CEI 611311.

Contenu de ce chapitre

Ce chapitre contient les sujets suivants :

Informations générales sur le langage à contacts (LD)

Présentation

La présente section décrit le langage à contacts (diagramme Ladder) LD selon CEI 61131-3.

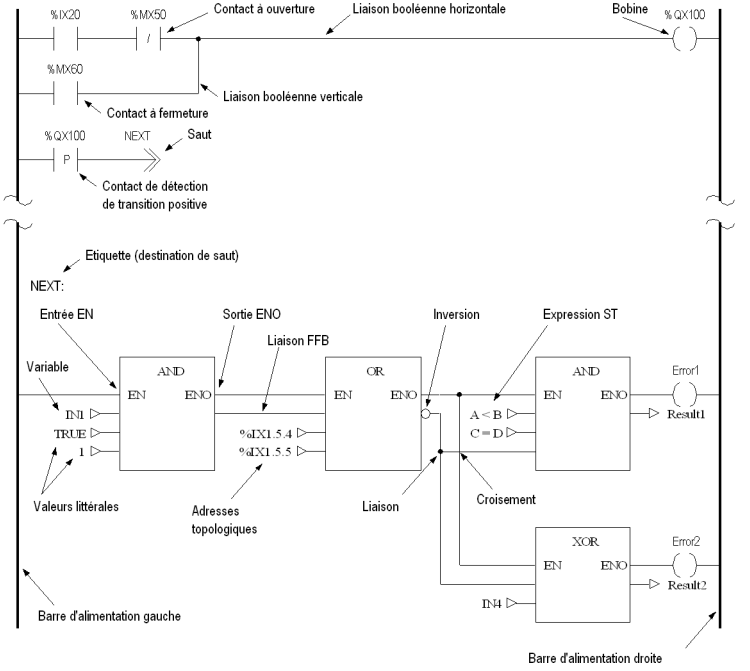
La structure d'une section LD correspond à un rung pour des montages à relais.

Sur le côté gauche de l'éditeur LD, se trouve la barre d'alimentation gauche. Cette barre d'alimentation gauche correspond à la phase (conducteur L) d'un rung. De même que sur un rung, le système ne "traite", lors de la programmation LD, que les objets LD qui sont branchés sur l'alimentation, c'est-à-dire qui sont reliés à la barre d'alimentation gauche. La barre d'alimentation droite correspond au conducteur neutre. Toutes les bobines et sorties FFB y sont reliées directement ou indirectement, ce qui permet d'établir un flux de courant.

Un groupe d'objets reliés les uns aux autres et ne présentant aucune liaison vers d'autres objets (à l'exception de la barre d'alimentation) est appelé réseau ou rung.

Représentation d'une section LD

Représentation :



Objets

Les objets du langage LD offrent des aides permettant de structurer une section en un ensemble de :

- Contacts *(voir page 358)*
- Bobines *(voir page 359)*
- EF et EFB (fonctions élémentaires *(voir page 361)* et blocs fonction élémentaires *(voir page 362)*),
- DFB (blocs fonction dérivés *(voir page 363)*),
- procédures ; *(voir page 363)*
- contrôles *(voir page 372)* et
- blocs d'opération et de comparaison *(voir page 374)* représentant une extension de la norme CEI 61131-3.

Ces objets peuvent être liés les uns aux autres par :

- des liaisons *(voir page 376)* ou
- des paramètres réels *(voir page 364)* (FFB uniquement).

La logique de la section peut être commentée par des objets texte (voir *Objet texte, page 380*).

Taille de la section

Une section LD comprend une fenêtre incluant une seule page.

Cette page est placée sur une grille qui partage la section en lignes et colonnes.

Les sections LD peuvent comporter de 11 à 64 colonnes et de 17 à 2000 lignes.

Le langage LD est basé sur les cellules, c'est-à-dire que seul un objet peut être placé dans chaque cellule.

Ordre d'exécution

L'ordre d'exécution des différents objets dans une section LD est déterminé par le flux de données à l'intérieur de la section. Les réseaux branchés sur la barre d'alimentation gauche sont traités de haut en bas (liaison avec la barre d'alimentation gauche). Les réseaux indépendants les uns des autres à l'intérieur de la section sont traités dans l'ordre de placement (de haut en bas) (voir également *Ordre d'exécution et flux de signaux, page 390*).

Conformité CEI

Pour plus d'informations sur la conformité CEI du langage LD, voir Conformité CEI *(voir page 657)*.

Contacts

Présentation

Un contact est un élément LD permettant de transférer un état de la liaison horizontale vers la droite. Cet état est le résultat d'une opération booléenne AND sur l'état de la liaison horizontale de gauche avec l'état du paramètre booléen réel associé.

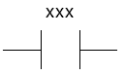
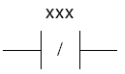
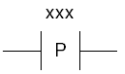
Un contact ne modifie pas la valeur du paramètre réel associé.

Les contacts occupent une cellule.

Sont autorisés comme paramètres réels :
- Variables booléennes
- Constantes booléennes
- Adresses booléennes (adresses topologiques ou symboliques)
- Expression ST *(voir page 509)* avec résultat booléen (par ex. `VarA OR VarB`)
  Les expressions ST comme paramètres réels de contacts représentent une extension de la norme CEI 61131-3 et doivent être activées de manière explicite.

Types de contacts

Les contacts disponibles sont les suivants :

Désignation	Représentation	Description
A fermeture		Dans le cas de contacts à fermeture, l'état de la liaison de gauche est transféré vers la liaison de droite si l'état du paramètre booléen réel associé (indiqué par xxx) est ON. Sinon, l'état de la liaison de droite est OFF.
A ouverture		Dans le cas de contacts à ouverture, l'état de la liaison de gauche est transféré vers la liaison de droite si l'état du paramètre booléen réel approprié (indiqué par xxx) est OFF. Sinon, l'état de la liaison de droite est OFF.
Contact de détection de transitions positives		Dans le cas de contacts de détection de transitions positives, la liaison de droite est ON pour un cycle de programme, si un passage de OFF à ON du paramètre réel booléen (xxx) associé a lieu et qu'en même temps, l'état de la liaison de gauche est ON. Sinon, l'état de la liaison de droite est 0. Voir aussi *Reconnaissance de front, page 381*.
Contact de détection de transitions négatives		Dans le cas de contacts de détection de transitions négatives, la liaison de droite est ON pour un cycle de programme, si un passage de ON à OFF du paramètre réel booléen (xxx) associé a lieu et qu'en même temps, l'état de la liaison de gauche est ON. Sinon, l'état de la liaison de droite est 0. Voir aussi *Reconnaissance de front, page 381*.

Bobines

Présentation

Une bobine est un élément LD permettant de transférer l'état de la liaison horizontale sur la gauche, inchangée, vers la liaison horizontale sur la droite. L'état est stocké dans le paramètre booléen réel respectif.

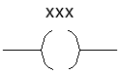
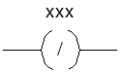
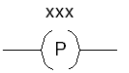
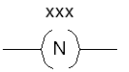
Normalement, les bobines suivent des contacts ou des FFB, mais elles peuvent aussi être suivies par des contacts.

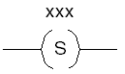
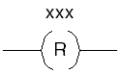
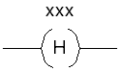
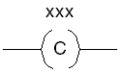
Les bobines occupent une cellule.

Sont autorisés comme paramètres réels :
- Variables booléennes
- Adresses booléennes (adresses topologiques ou symboliques)

Types de bobines

Les bobines suivantes sont disponibles :

Désignation	Représentation	Description
Bobine		Dans le cas de bobines, l'état de la liaison de gauche est transféré vers le paramètre booléen réel associé (indiqué par xxx) et la liaison de droite.
bobine inverse		Dans le cas de bobines inverses, l'état de la liaison de gauche est copié sur la liaison de droite. L'état inversé de la liaison de gauche est copié vers le paramètre booléen réel associé (indiqué par xxx). Si la liaison de gauche est OFF, alors la liaison de droite sera également OFF et le paramètre booléen réel associé sera ON.
Bobine de détection de transitions positives		Dans le cas de bobines de détection de transitions positives, l'état de la liaison de gauche est copié sur la liaison de droite. Le paramètre réel associé du type de données EBOOL (indiqué par xxx) est 1 pour un cycle de programme, si un passage de 0 à 1 de la liaison gauche est effectué. Voir aussi *Reconnaissance de front, page 381*.
Bobine de détection de transitions négatives		Dans le cas de bobines de détection de transitions négatives, l'état de la liaison de gauche copié sur la liaison de droite. Le paramètre réel booléen associé (indiqué par xxx) est 1 pour un cycle de programme, si un passage de 1 à 0 de la liaison gauche est effectué. Voir aussi *Reconnaissance de front, page 381*.

Désignation	Représentation	Description
Bobine d'enclenchement		Avec une bobine d'enclenchement, l'état de la liaison de gauche est copié sur la liaison de droite. Le paramètre booléen réel associé (indiqué par xxx) est défini sur ON si l'état de la liaison de gauche est ON, sinon il reste inchangé. Le paramètre booléen réel associé peut être réinitialisé via la bobine de réinitialisation. Voir aussi *Reconnaissance de front, page 381*.
Bobine de réinitialisation		Avec une bobine de réinitialisation, l'état de la liaison de gauche est copié sur la liaison de droite. Le paramètre booléen réel associé (indiqué par xxx) est défini sur OFF si l'état de la liaison de gauche est ON, sinon il reste inchangé. Le paramètre booléen réel associé peut être enclenché via la bobine d'enclenchement. Voir aussi *Reconnaissance de front, page 381*.
Bobine d'arrêt		Avec des bobines d'arrêt, si le statut de la liaison de gauche est 1, l'exécution du programme est arrêtée immédiatement. (Avec des bobines d'arrêt, l'état de la liaison de gauche n'est pas copié sur la liaison de droite.)
Bobine d'appel		Avec des bobines d'appel, l'état de la liaison de gauche est copié vers la liaison de droite. Si l'état de la liaison de gauche est ON alors le sous-programme associé (indiqué par xxx) est appelé. Le sous-programme à appeler doit se trouver dans la même tâche que la section LD appelante. Il est possible d'appeler des sous-programmes au sein de sous-programmes. Les sous-programmes sont un complément de la norme CEI 61131-3 et doivent être activés de manière explicite. Dans les sections d'actions SFC, les bobines d'appel (appels de sous-programmes) ne sont autorisés que si le mode Multitoken a été activé.

Fonctions élémentaires, blocs fonction élémentaires, blocs fonction dérivés et procédures (FFB)

Introduction

FFB est le terme générique pour :
- les fonctions élémentaires (EF) *(voir page 361)*
- les blocs fonction élémentaires (EFB) *(voir page 362)*
- les blocs fonction dérivés (DFB) *(voir page 363)*
- Procédure *(voir page 363)*

Les FFB occupent une largeur de 1 à 3 colonnes (en fonction de la longueur des noms des paramètres formels) et une longueur de 2 à 33 lignes (en fonction du nombre de lignes des paramètres formels).

Fonction élémentaire

Les fonctions n'ont pas d'état interne. Lorsque les valeurs d'entrée sont identiques, la valeur de sortie est la même à chaque exécution de la fonction. Par exemple, l'addition de deux valeurs donne toujours le même résultat.

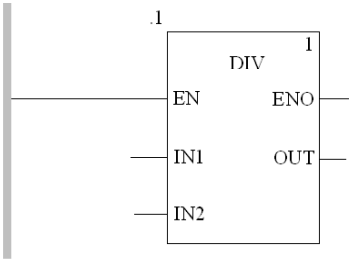
Une fonction élémentaire est représentée graphiquement comme un cadre avec des entrées et une sortie. Les entrées sont toujours représentées sur la gauche et la sortie toujours sur la droite du cadre.

Le nom de la fonction, c'est-à-dire le type de fonction, est affiché au centre du cadre.

Le numéro d'exécution *(voir page 390)* de la fonction apparaît à droite du type de fonction.

Le numéro de fonction est affiché au-dessus du cadre. Le numéro de fonction représente le numéro courant de la fonction dans la section actuelle. Les numéros de fonction ne peuvent pas être modifiés.

Fonction élémentaire



Pour certaines fonctions élémentaires, il est possible d'augmenter le nombre d'entrées.

Bloc fonction élémentaire

Les blocs fonction élémentaires ont des états internes. Lorsque les valeurs d'entrée sont identiques, la valeur de sortie peut être différente pour toutes les exécutions de la fonction. Par exemple, pour un compteur, la valeur de sortie augmente.

Un bloc fonction élémentaire est représenté graphiquement sous forme de cadre avec des entrées et des sorties. Les entrées sont toujours représentées sur la gauche et les sorties toujours sur la droite du cadre. Le nom du bloc fonction, c'est-à-dire le type de bloc fonction, est affiché au centre du cadre. Le nom d'instance est affiché au-dessus du cadre.

Les blocs fonction peuvent avoir plusieurs sorties.

Le nom du bloc fonction, c'est-à-dire le type de bloc fonction, est affiché au centre du cadre.

Le numéro d'exécution *(voir page 390)* du bloc fonction apparaît à droite du type de bloc fonction.

Le nom d'instance est affiché au-dessus du cadre.

Le nom d'instance permet d'identifier précisément le bloc fonction dans un projet.

Le nom d'instance est généré automatiquement et présente la structure suivante : `FBI_n`

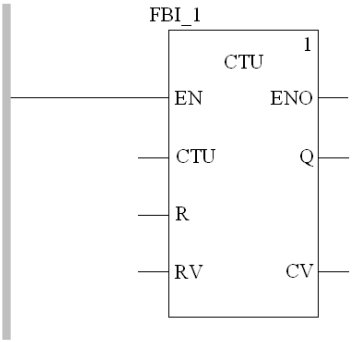
`FBI` = Instance de bloc fonction

`n` = numéro courant du bloc fonction au sein du projet

Vous pouvez modifier ces noms générés automatiquement pour rendre la vue d'ensemble plus claire. Le nom d'instance (32 caractères maximum) doit être unique dans tout le projet ; aucune distinction n'est faite ici entre majuscules et minuscules. Le nom d'instance doit respecter les conventions de noms générales.

NOTE : conformément à la norme CEI 61131-3, les noms d'instance doivent commencer par une lettre. Si vous voulez également utiliser des chiffres comme premier caractère, vous devez activer cette fonction.

Bloc fonction élémentaire

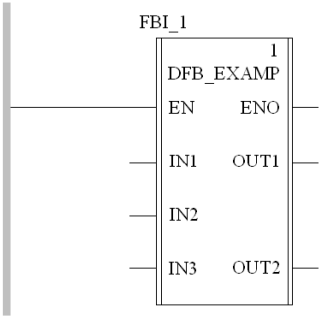


DFB

Les blocs fonction dérivés (DFB) ont les mêmes caractéristiques que les blocs fonction élémentaires. Ils sont cependant créés par l'utilisateur dans les langages FBD, LD, IL et/ou ST.

L'unique différence par rapport aux blocs fonction élémentaires est que le bloc fonction dérivé est représenté graphiquement sous forme de cadre avec deux lignes verticales.

Bloc fonction dérivé



Procédure

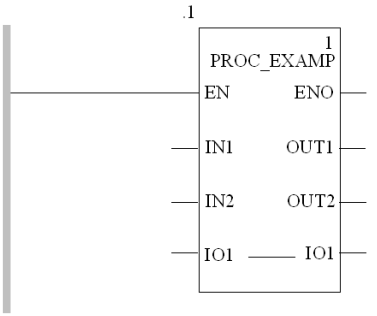
Techniquement, les procédures sont des fonctions.

L'unique différence par rapport aux fonctions élémentaires est que les procédures peuvent comprendre plus d'une sortie et qu'elles prennent en charge le type de données `VAR_IN_OUT`.

Il n'y a pas de différence physique entre les procédures et les fonctions élémentaires.

Les procédures sont une extension de la norme CEI 61131-3 et doivent être activées de manière explicite.

Procédure

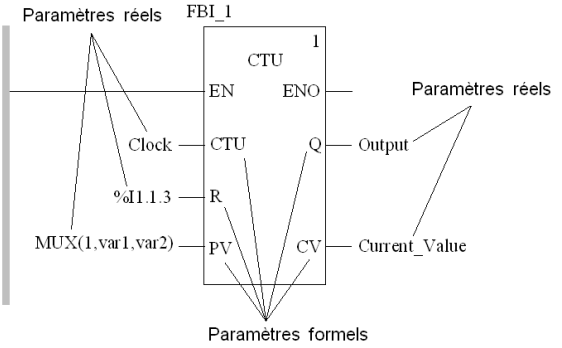


Paramètres

Pour importer des valeurs dans le FFB ou exporter des valeurs du FFB, des entrées et des sorties sont nécessaires. Elles sont appelées paramètres formels.

Les paramètres formels sont liés à des objets qui comprennent les états courants du traitement. Ces états sont appelés paramètres réels.

Paramètres formels et réels :



Durant l'exécution du programme, les valeurs sont transmises, par le biais des paramètres réels, du processus au FFB, et renvoyées à nouveau à la sortie après le traitement.

Seul un objet (paramètre réel) du type de données suivant peut être relié aux entrées FFB :

- contact
- variable
- adresse
- libellé
- expression ST
  Les expressions ST des entrées FFB représentent une extension de la norme CEI 61131-3 et doivent être activées de manière explicite.
- Liaison

Les combinaisons d'objets (paramètres réels) suivantes peuvent être reliées aux sorties FFB :

- une ou plusieurs bobines
- un ou plusieurs contacts
- une variable
- une variable et une ou plusieurs liaisons (non valable pour les sorties VAR_IN_OUT *(voir page 371)*)
- une adresse,
- une adresse et une ou plusieurs liaisons (non valable pour les sorties VAR_IN_OUT *(voir page 371)*)
- une ou plusieurs liaisons (non valable pour les sorties VAR_IN_OUT *(voir page 371)*)

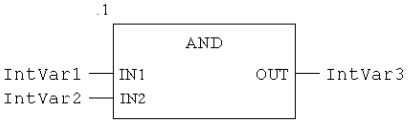
Le type des données de l'objet à relier doit correspondre au type des données de l'entrée/la sortie FFB. On choisira un type de données adapté pour le bloc fonction, si tous les paramètres réels sont constitués de valeurs littérales.

Exception : pour les entrées/sorties génériques FFB de type de données ANY_BIT, des objets de type de données INT ou DINT (pas UINT ni UDINT) peuvent être reliés.

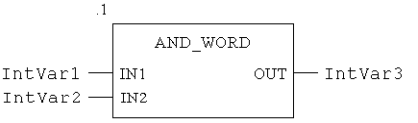
Il s'agit d'une extension de la norme CEI 61131-3 et doit donc être activée de manière explicite.

Exemple :

Autorisé :



Non autorisé :



(Dans ce cas AND_INT doit être utilisé.)

Il n'est en principe pas nécessaire d'affecter un paramètre réel à tous les paramètres formels. Cependant, cela n'est pas valable pour les broches inversées. Un paramètre réel doit toujours leur être affecté. Cela est également impératif pour certains types de paramètre formel. Pour connaître les types concernés, veuillez vous reporter au tableau suivant.

Tableau des types de paramètre formel :

Type de paramètre	EDT	STRING	ARRAY	ANY_ARRAY	IODDT	STRUCT	FB	ANY
EFB : Entrée	-	+	+	+	/	+	/	+
DFB : Sortie	-	-	+	/	/	-	/	+
EFB : VAR_IN_OUT	+	+	+	+	+	+	/	+
DFB : Entrée	-	+	+	+	/	+	/	+
DFB : VAR_IN_OUT	+	+	+	+	+	+	/	+
EFB : Sortie	-	-	+	+	+	-	/	+
EF : Entrée	-	-	+	+	+	+	+	+
EF : VAR_IN_OUT	+	+	+	+	+	+	/	+

Type de paramètre	EDT	STRING	ARRAY	ANY_ARRAY	IODDT	STRUCT	FB	ANY
EF : Sortie	-	-	-	-	-	-	/	-
Procédure : Entrée	-	-	+	+	+	+	+	+
Procédure : VAR_IN_OUT	+	+	+	+	+	+	/	+
Procédure : Sortie	-	-	-	-	-	-	/	+
+ paramètre réel impératif								
- Paramètre réel non impératif								
/ Non applicable								

Les FFB utilisant aux entrées des paramètres réels, auxquels aucune valeur n'a encore été affectée, fonctionnent avec les valeurs initiales de ces paramètres réels.

Si aucune valeur n'est affectée à un paramètre formel, la valeur initiale est utilisée pendant l'exécution du bloc fonction. Si aucune valeur initiale n'est définie, la valeur par défaut (0) est utilisée.

Si aucune valeur n'est affectée à un paramètre formel et que le bloc fonction/DFB a été instancié à plusieurs reprises, les instances appelées par la suite travaillent avec l'ancienne valeur.

Variables publiques

Certains blocs fonction disposent non seulement d'entrées et de sorties, mais également de variables publiques (Public Variables).

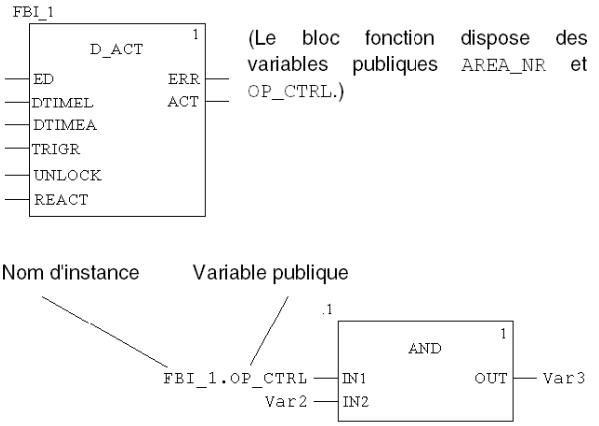
Ces variables permettent de transmettre des valeurs statiques (valeurs non influencées par le procédé) au bloc fonction. Elles sont donc utilisées lors du paramétrage du bloc fonction.

Les variables publiques sont une extension de la norme CEI 61131-3.

Les valeurs sont affectées aux variables publiques via leur valeur initiale.

Les valeurs des variables publiques sont ensuite lues à partir du nom d'instance du bloc fonction et du nom de la variable publique.

Exemple :



Variables privées

Certains blocs fonction disposent non seulement d'entrées, de sorties et de variables publiques, mais également de variables privées.

A l'instar des variables publiques, ces variables permettent de transmettre des valeurs statistiques (valeurs non influencées par le procédé) au bloc fonction.

Le programme utilisateur ne peut pas accéder à ces variables. Seule la table d'animation en a la capacité.

NOTE : les DFB imbriqués sont déclarés comme des variables privées du DFB parent. Ainsi, leurs variables ne sont pas accessibles via la programmation, mais via la table d'animation.

Les variables privées sont une extension de la norme CEI 61131-3.

Remarques sur la programmation

Veuillez tenir compte des remarques qui suivent sur la programmation :

- Les FFB ne sont traités que lorsqu'ils sont connectés directement ou indirectement à la barre d'alimentation gauche.
- Si le FFB doit être exécuté de façon conditionnelle, l'entrée EN peut être préalablement reliée par des contacts ou d'autres FFB (voir également EN et ENO *(voir page 368)*).
- Les entrées et sorties booléennes peuvent être inversées.
- Des conditions particulières s'appliquent lors de l'utilisation de variables VAR_IN_OUT *(voir page 371)*.
- Les instances de DFB ou bloc fonction peuvent être appelées à plusieurs reprises (voir aussi Appel multiple d'une instance de bloc fonction *(voir page 368)*).

Appel multiple d'une instance de bloc fonction

Les instances de DFB ou bloc fonction peuvent être appelées à plusieurs reprises, à l'exception des instances d'EFB de communication et blocs fonction/DFB ayant une sortie ANY mais pas d'entrée ANY, qui ne peuvent être appelées qu'une seule fois.

L'appel multiple de la même instance de DFB/bloc fonction est par exemple utile dans les cas suivants :

- si le bloc fonction/DFB ne comporte aucune valeur interne ou si celle-ci n'est plus nécessaire pour un traitement ultérieur.
  Dans ce cas, l'appel multiple de la même instance de DFB/bloc fonction permet d'économiser de l'espace mémoire, car le code du bloc fonction/DFB n'est alors chargé qu'une seule fois.
  Le bloc fonction/DFB est pour ainsi dire traité comme une "fonction".
- si le bloc fonction/DFB comprend des valeurs internes et que celles-ci doivent être influencées à différents endroits du programme, la valeur d'un compteur, par exemple, doit être augmentée à différents endroits du programme.
  Dans ce cas, l'appel multiple de la même instance de bloc fonction/DFB permet d'économiser la mémoire des résultats intermédiaires pour un traitement ultérieur à un autre endroit du programme.

EN et ENO

Une entrée EN et une sortie ENO peuvent être configurées pour tous les FFB.

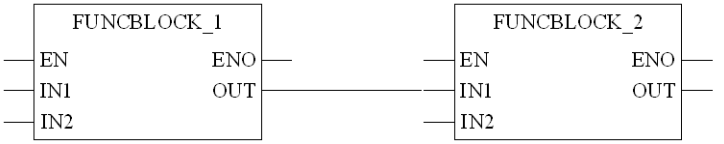
Si la valeur de EN est déjà à "0", lors de l'appel de FFB, les algorithmes définis par FFB ne sont pas exécutés et ENO est réglé sur 0.

Si la valeur de EN est déjà à "1", lors de l'appel de FFB, les algorithmes définis par FFB sont exécutés. Après l'exécution sans erreur de ces algorithmes, la valeur de ENO est mise à "1". Si une erreur se produit durant l'exécution de ces algorithmes, ENO est réglé sur 0.

Si aucune valeur n'est attribuée à la broche EN à l'appel du FFB, l'algorithme défini par ce dernier est exécuté (comme lorsque EN a la valeur « 1 »). Reportez-vous à la section *Maintenir les liens de sortie sur les EF désactivés (voir Unity Pro, Modes de marche,)*.

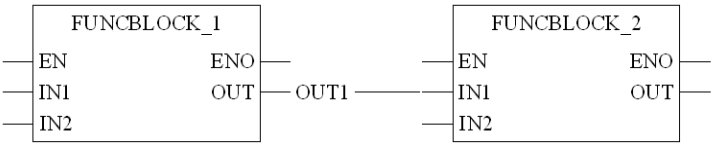
Si ENO est réglé sur 0 (car EN = 0 ou en raison d'une erreur d'exécution) :

- Blocs fonction
  - Traitement EN/ENO pour les blocs fonction ayant (seulement) une liaison comme paramètre de sortie :



Lorsque EN est réglé sur 0 par FUNCBLOCK_1, la liaison à la sortie OUT de FUNCBLOCK_1 conserve l'ancien statut qu'elle avait lors du dernier cycle correct.

- Traitement EN/ENO pour les blocs fonction ayant une variable et une liaison comme paramètre de sortie :

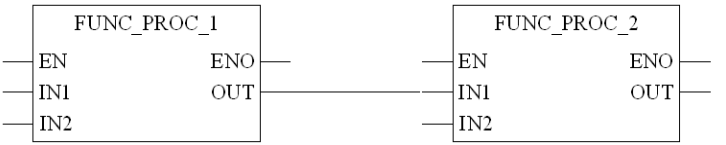


Lorsque EN est réglé sur 0 par FUNCBLOCK_1, la liaison à la sortie OUT de FUNCBLOCK_1 conserve l'ancien statut qu'elle avait lors du dernier cycle correct. La variable OUT1 située sur la même broche conserve son ancien statut ou peut être modifiée depuis l'extérieur sans avoir d'influence sur la liaison. La variable et la liaison sont enregistrées indépendamment l'une de l'autre.

- Fonctions/Procédures

  Selon la définition CEI 61131-3, les sorties de fonctions désactivées (entrée EN mise à "0") sont indéfinies. (Le même principe s'applique aux procédures.) Voici, néanmoins, une explication des états des sorties dans un tel cas :

  - Traitement EN/ENO pour les fonctions/procédures ayant (seulement) une liaison comme paramètre de sortie :



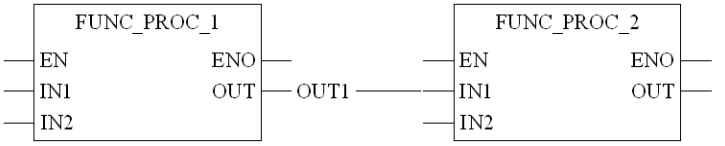
Si EN de FUNC_PROC_1 est réglé sur 0, la valeur de la liaison sur la sortie OUT de FUNC_PROC_1 dépend du paramètre de projet **Maintenir les liens de sortie sur les EF désactivés**, disponible depuis Unity Pro 4.1.

Si ce paramètre de projet est réglé sur 0, la valeur de la liaison est réglée sur 0.

Si ce paramètre de projet est réglé sur 1, la liaison conserve la valeur qu'elle avait lors du dernier cycle exécuté correctement.

Pour plus d'informations, consultez la section *Maintenir les liens de sortie sur les EF désactivés (voir Unity Pro, Modes de marche,)*.

- Traitement EN/ENO pour les fonctions/procédures ayant une variable et une liaison comme paramètre de sortie :



Si EN de FUNC_PROC_1 est réglé sur 0, la valeur de la liaison sur la sortie OUT de FUNC_PROC_1 dépend du paramètre de projet **Maintenir les liens de sortie sur les EF désactivés**, disponible depuis Unity Pro 4.1.
Si ce paramètre de projet est réglé sur 0, la valeur de la liaison est réglée sur 0.
Si ce paramètre de projet est réglé sur 1, la liaison conserve la valeur qu'elle avait lors du dernier cycle exécuté correctement.
Pour plus d'informations, consultez la section *Maintenir les liens de sortie sur les EF désactivés (voir Unity Pro, Modes de marche,)*.
La variable OUT1 située sur la même broche conserve son ancien statut ou peut être modifiée depuis l'extérieur sans avoir d'influence sur la liaison. La variable et la liaison sont enregistrées indépendamment l'une de l'autre.

Le comportement aux sorties des FFB est indépendant du fait que les FFB soient appelés sans EN/ENO ou avec EN = 1.

NOTE : pour les blocs fonction désactivés (EN = 0) équipés d'une fonction d'horloge interne (par exemple, le bloc fonction DELAY), le temps semble continuer de s'écouler, car il est calculé à l'aide d'une horloge système et est, par conséquent, indépendant du cycle du programme et de la libération du bloc.

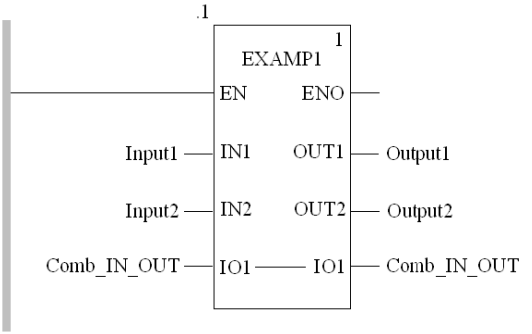
Variable VAR_IN_OUT

Les FFB sont souvent utilisés pour lire une variable à l'entrée (variables d'entrée), pour la traiter et pour transmettre de nouveau les valeurs modifiées de cette **même** variable (variables de sortie).

Ce cas exceptionnel d'une variable d'entrée/de sortie est également appelé variable VAR_IN_OUT.

Dans le FFB, une ligne indique que les variables d'entrée et de sortie sont liées l'une à l'autre.

Variable VAR_IN_OUT



Il convient de noter les particularités suivantes en cas d'utilisation de FFB avec des variables VAR_IN_OUT :

- une variable doit être affectée à toutes les entrées VAR_IN_OUT.
- les liaisons graphiques permettent uniquement de relier des sorties VAR_IN_OUT à des entrées VAR_IN_OUT.
- seule une liaison graphique unique peut être reliée à une entrée/sortie VAR_IN_OUT.
- une combinaison de variables/d'adresses et de liaisons graphiques n'est pas possible pour les sorties VAR_IN_OUT.
- il est interdit de relier des valeurs littérales ou des constantes à des entrées/sorties VAR_IN_OUT.
- il est interdit d'utiliser des négations au niveau des entrées/sorties VAR_IN_OUT.
- des variables/composantes de variables différentes peuvent être reliées à l'entrée VAR_IN_OUT et à la sortie VAR_IN_OUT. Dans un tel cas, la valeur de la variable/composante de variable à l'entrée est copiée dans la variable/composante de variable à la sortie.

Contrôles

Introduction

Les éléments de commande servent à l'exécution de sauts au sein d'une section LD et au retour prématuré dans le programme principal depuis un sous-programme (SRx) ou un bloc fonction dérivé (DFB).

Les éléments de commande occupent une cellule.

Contrôles

Les contrôles suivants sont disponibles.

Désignation	Représentation	Description
Jump		Si l'état de la liaison gauche est 1, un saut est exécuté jusqu'à l'étiquette (dans la section courante). Pour générer un saut inconditionnel, l'objet saut est placé directement sur la barre d'alimentation gauche. Pour générer un saut conditionnel, l'objet saut est placé à la fin d'une rangée de contacts.
Libellé	`LABEL:`	Les repères (destinations de saut) sont représentés comme du texte avec deux-points à la fin. Le texte est limité à 32 caractères et doit être unique dans l'ensemble de la section. Le texte doit respecter les conventions de nommage générales. Les étiquettes de saut ne peuvent être placées que dans la première cellule directement sur la barre d'alimentation gauche. **Note :** Les étiquettes de saut ne doivent "couper" aucun réseau, c'est-à-dire qu'une ligne imaginaire entre l'étiquette de saut et la marge droite de la section ne doit être coupée par aucun objet. Cela s'applique également aux liaisons booléennes et FFB.

Désignation	Représentation	Description

Return		Des objets RETURN ne peuvent pas être utilisés dans le programme principal.
		● Dans un DFB, un objet RETURN force le retour au programme qui a appelé le DFB.
		  ● Le reste de la section de DFB contenant l'objet RETURN n'est pas exécuté.
		  ● Les sections suivantes du DFB ne sont pas exécutées.
		Le programme qui a appelé le DFB sera exécuté après retour du DFB.
		Si le DFB est appelé par un autre DFB, le DFB appelant sera exécuté après le retour.
		● Dans un SR, un objet RETURN force le retour au programme qui a appelé le SR.
		  ● Le reste du SR contenant l'objet RETURN n'est pas exécuté.
		Le programme qui a appelé le SR sera exécuté après retour du SR.

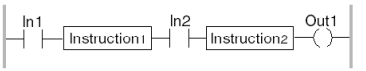
Blocs opération et blocs comparaison

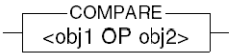
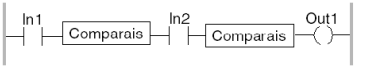
Introduction

En plus des objets définis dans la norme CEI 61131-3, il existe d'autres blocs servant à l'exécution d'instructions ST *(voir page 509)* et d'expressions ST *(voir page 509)* et à des opérations de comparaison simples. Ces blocs sont exclusivement disponibles dans le langage de programmation LD.

Objets

Les objets suivants sont disponibles :

Désignation	Représentation	Description
Bloc opération		Si l'état de la liaison gauche est 1, l'instruction ST comprise dans le bloc est exécutée. Toutes les instructions ST *(voir page 509)* sont permises **sauf** les instructions de commande : • (`RETURN`, • `JUMP`, `IF`, • `CASE`, • `FOR` • etc.) Pour les blocs opération, quel que soit le résultat de l'instruction ST, l'état de la liaison gauche est transmis à la liaison droite. Un bloc peut contenir jusqu'à 4 096 caractères. Si tous les caractères ne peuvent pas être affichés, les premiers caractères seront affichés suivis de points de suspension (...). Un bloc opération occupe 1 ligne et 4 colonnes. Exemple :  Dans l'exemple, `Instruction1` est exécutée si `In1`=1. `Instruction2` est exécutée si `In1`=1 et `In2`=1 (le résultat de `Instruction1` ne joue aucun rôle pour l'exécution de `Instruction2`). Out1 est 1, si `In1`=1 et `In2`=1 (les résultats de `Instruction1` et `Instruction2` n'influent pas sur l'état de `Out1`).

Désignation	Représentation	Description
Bloc de comparaison horizontal		Les blocs de comparaison horizontaux servent à exécuter une expression de comparaison (<, >, <=, >=, =, <>) dans le langage de programmation ST. (Remarque : La même fonctionnalité est également disponible via les expressions ST *(voir page 509)*.) Un bloc comparaison exécute un ET de sa broche d'entrée gauche et du résultat de sa condition de comparaison, puis affecte le résultat du ET de façon inconditionnelle à la broche de sortie droite. Par exemple, lorsque l'état de la liaison gauche est 1 et que le résultat de la comparaison est 1, l'état de la liaison droite est 1. Un bloc de comparaison horizontal peut contenir jusqu'à 4 096 caractères. Si tous les caractères ne peuvent pas être affichés, les premiers caractères seront affichés suivis de points de suspension (...). Un bloc de comparaison horizontal occupe une ligne et deux colonnes. Exemple :  Dans l'exemple, `Comparaison1` est exécutée si `In1`=1. `Comparaison2` est exécutée si `In1`=1 , `In2`=1, résultat de `Comparaison1`=1. Out1 est 1, si`In1`=1, `In2`=1, le résultat de `Comparaison1`=1 et le résultat de `Comparaison2`=1.

Liaisons

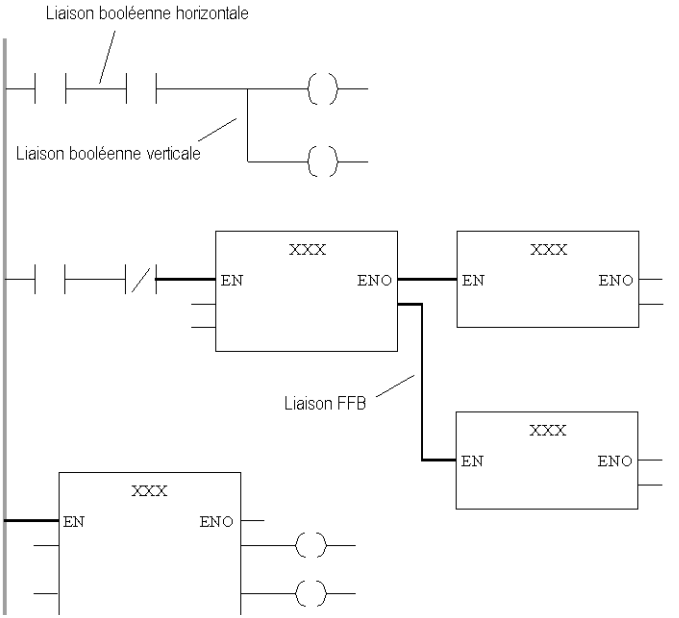
Description

Les liaisons sont des liens entre des objets LD (contacts, bobines, FFB, etc.).

Une différence est faite entre deux types de liaison.

- Liaison booléenne
  Les liaisons booléennes comprennent un ou plusieurs segments qui relient entre eux des objets booléens (contacts, bobines).
  Pour les liaisons booléennes, une différence est faite entre :
  - Les liaisons booléennes horizontales
    Les liaisons booléennes horizontales permettent une liaison en série de contacts et bobines.
  - Les liaisons booléennes verticales
    Les liaisons booléennes verticales permettent une liaison en parallèle de contacts et bobines.
- Liaisons FFB
  Les liaisons FFB comprennent une combinaison de segments horizontaux et verticaux qui relient les entrées/sorties FFB avec d'autres objets.

Liaisons :



Remarques générales sur la programmation

Veuillez observer les remarques générales qui suivent sur la programmation :

- Les types de données respectifs des entrées/sorties à relier doivent correspondre les uns aux autres.
- Les liaisons entre des paramètres de longueur variable (ex : `ANY_ARRAY_INT`) ne sont pas permises.
- Plusieurs liaisons peuvent être reliées à une entrée (côté droit d'un contact/d'une bobine, sortie FFB). Cependant une seule liaison est possible avec chaque entrée (côté gauche d'un contact/d'une bobine, entrée FFB).
- Les contacts, bobines et entrées de FFB non reliés obtiennent par défaut la valeur "0".
- Les boucles ne peuvent pas être configurées par le biais de liaisons, étant donné que, dans ce cas, l'ordre d'exécution dans la section ne peut pas être défini de façon unique. Les boucles doivent être résolues par le biais de paramètres réels (voir *Boucles non permises, page 392*).

Remarques sur la programmation de liaisons booléennes

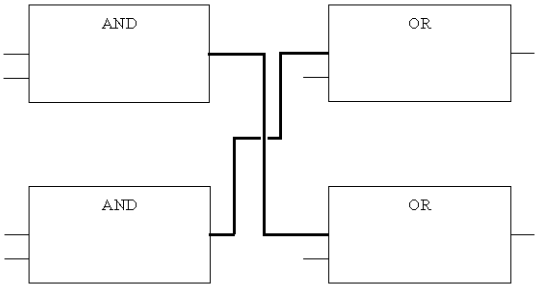
Remarques sur la programmation de liaisons booléennes :

- Le chevauchement des liaisons booléennes avec d'autres objets **n'est pas** admis.
- Le flux de signaux (passage de courant) d'une liaison booléenne va de gauche vers à droite. C'est pourquoi les liaisons dirigées vers l'arrière ne sont pas autorisées.
- Si deux liaisons booléennes se croisent, un lien entre les deux liaisons est automatiquement créé. Etant donné que le croisement de liaisons booléennes n'est pas possible, les liaisons ne sont pas identifiées de manière particulière.

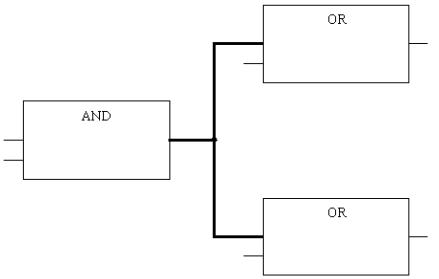
Remarques sur la programmation des liaisons FFB

Remarques sur la programmation des liaisons FFB :

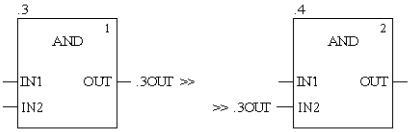
- Au moins un côté d'une liaison FFB doit être relié à une entrée ou une sortie FFB.
- Les liaisons FFB sont représentées par un trait double afin de les différencier des liaisons booléennes.
- Le flux de signaux (passage de courant) d'une liaison FFB va de la sortie FFB vers l'entrée FFB, indépendamment de la direction. C'est pourquoi les liaisons dirigées vers l'arrière sont autorisées.
- Seules des entrées FFB et des sorties FFB peuvent être reliées ensemble. La liaison de plusieurs sorties FFB entre elles n'est pas possible. Cela signifie qu'en LD aucune liaison OU n'est possible via des liaisons FFB.
- Le chevauchement des liaisons FFB avec d'autres objets est admis.
- Le croisement de liaisons FFB est admis. Les croisements sont représentés par une liaison interrompue.



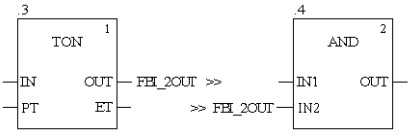
- Dans le cadre de liaisons FFB, les points de liaison entre plusieurs liaisons FFB sont représentés par un cercle rempli.



- Afin d'éviter le croisement de liaisons, les liaisons FFB peuvent également être représentées sous forme de connecteurs.
  A cette occasion, la source et la cible de la liaison FFB sont caractérisées par un nom unique au sein de la section.
  Suivant le type d'objet source, le nom du connecteur est formé comme suit :
  - pour les fonctions : " numéro de fonction/paramètre formel " de la source de la liaison.



  - pour les blocs fonction : " nom d'instance/paramètre formel " de la source de la liaison.



- Pour les contacts : "OUT1_numéro courant".

Liaisons verticales

La "liaison verticale" constitue un cas particulier de liaison. La liaison verticale tient lieu de OU logique. Avec cette forme de liaison OU, 32 entrées (contacts) et 64 sorties (bobines, liaisons) sont possibles.

Objet texte

Présentation

Dans le plan de contacts LD, les textes peuvent être placés sous forme d'objets texte. La taille de ces objets texte est fonction de la longueur du texte. Selon la longueur du texte, la taille de l'objet peut être agrandie, dans les sens vertical et horizontal, d'unités de grille supplémentaires. Les objets texte peuvent chevaucher d'autres objets.

Reconnaissance de front

Introduction

Pendant la reconnaissance d'un front, un bit est surveillé pendant un passage de 0 à 1 (front positif ou ascendant) ou de 1 à 0 (front négatif ou descendant).

Pour ce faire, la valeur du bit du cycle précédent est comparée à la valeur du bit du cycle en cours. Dans ce cas, non seulement la valeur en cours est requise mais également l'ancienne valeur.

C'est pourquoi, lors de la détection de fronts, 2 bits (valeurs courante et ancienne) sont requis au lieu d'un.

Comme le type de données `BOOL` n'offre qu'un seul bit (valeur courante), un autre type de données est utilisé pour la détection de fronts, `EBOOL` (`BOOL` étendu). Outre la détection des fronts, le type de données `EBOOL` offre une option pour le forçage. Vous devez donc également l'enregistrer que le forçage de bit soit actif ou non.

Le type de données `EBOOL` enregistre les données suivantes :

- la valeur courante du bit dans le *bit de valeur*,
- la valeur précédente du bit dans le *bit d'historique*,
  (au début de chaque cycle, le contenu du bit de valeur est copié dans le bit d'historique)
- les informations sur l'activation du forçage du bit dans le *bit de forçage*,
  (0 = forçage inactif, 1 = forçage actif)

Restrictions d'EBOOL

⚠ ATTENTION

COMPORTEMENT IMPREVU DE L'EQUIPEMENT

Pour une bonne détection des fronts, `%M` doit être mis à jour à chaque cycle de tâche. Lors d'une écriture unique, le front est infini.

Le non-respect de ces instructions peut provoquer des blessures ou des dommages matériels.

En utilisant une variable EBOOL pour les contacts afin de reconnaître les fronts positifs (P) et négatifs (N) (fronts ascendants et descendants, avec une EF), vous devez respecter les restrictions ci-après.

EBOOL avec %M non écrit dans le programme

Une variable EBOOL avec une adresse %M qui n'est pas écrite dans le programme, mais fournie directement (par une table d'animation, un écran ou une interface, par exemple), fonctionnera de manière imprévue. La valeur du front reste indéfiniment TRUE car %M n'est écrit qu'une seule fois.

NOTE : pour éviter cela, %M doit être écrit à la fin de la tâche pour mettre à jour les informations de valeur anciennes.

L'ancienne valeur n'est mise à jour que lorsque le bit %M est écrit. Par conséquent, si vous n'écrivez le bit qu'une fois, la détection de front est infinie.

Valeur ancienne	Valeur actuelle	Détection de front	Description
0	0	0	état 0 (avant d'écrire le bit)
0	1	1	Ecrivez 1 dans le bit (avec une table d'animation, par exemple).
0	1	1	Si vous ne l'écrivez pas à nouveau, le front demeure indéfiniment.
1	1	0	Ecrivez à nouveau 1 dans le bit, la valeur ancienne est actualisée et la détection de front est réglée sur 0.

EBOOL avec %M écrit dans le programme

Pour une variable EBOOL avec une adresse %M écrite dans le programme, vous devez respecter les limitations suivantes :

- N'utilisez pas le bit avec une bobine d'enclenchement ou de déclenchement. Dans ce cas, l'ancienne valeur n'est pas actualisée. Vous pouvez donc établir un front infini.
- N'écrivez pas le bit de manière conditionnelle. Une logique simple telle que `IF NOT %M1 THEN %M1 := TRUE; END_IF` cause un front éternel car elle n'est écrite qu'une fois.

EBOOL avec %I

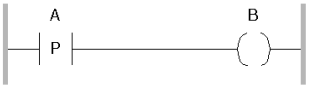
Pour une variable EBOOL avec une adresse %I, vous devez respecter les restrictions suivantes :

- En fonctionnement multitâche, la vérification du front %I doit être effectuée dans la tâche où il est actualisé. Il est déconseillé d'utiliser la détection du front d'un %I planifiée dans une tâche de priorité supérieure.
  Exemple : si vous avez une tâche fast qui actualise %I, n'utilisez pas la détection de front dans la tâche mast. Selon la planification, le front peut être détecté ou non.

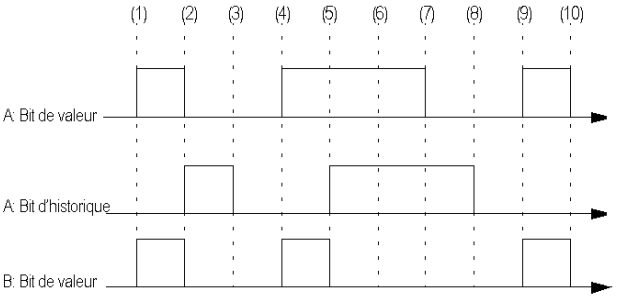
Détection de fronts positifs

Pour détecter des fronts positifs, vous devez utiliser un contact pour la détection de fronts positifs. Avec ce contact, la connexion de droite pour un cycle de programme est 1 lorsque la transition du paramètre associé réel (A) passe de 0 à 1 et que, simultanément, l'état de la connexion de gauche est 1. Sinon, l'état de la liaison de droite est 0.

Dans l'exemple, un front positif de la variable A doit être reconnu et B doit donc être défini pendant un cycle.



Chaque fois que le bit de valeur de A est égal à 1 et que le bit d'historique est égal à 0, B est réglé sur 1 pendant un cycle (cycle 1, 4 et 9).

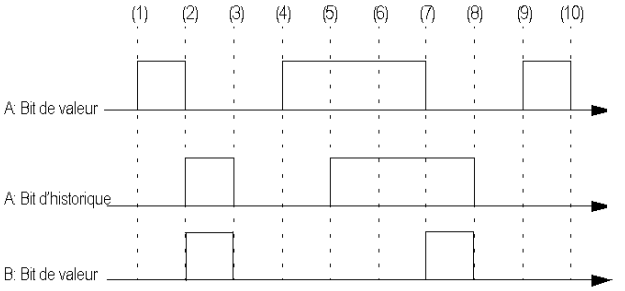


Détection de fronts négatifs

Pour détecter des fronts négatifs, vous devez utiliser un contact pour la détection de fronts négatifs. Avec ce contact, la connexion de droite pour un cycle de programme est 1 lorsque la transition du paramètre associé réel (A) passe de 1 à 0 et que, simultanément, l'état de la connexion de gauche est 1. Sinon, l'état de la liaison de droite est 0.

Dans l'exemple, un front négatif de la variable A doit être reconnu et B doit donc être défini pendant un cycle.

Chaque fois que le bit de valeur de A est égal à 0 et que le bit d'historique est égal à 1, B est réglé sur 1 pendant un cycle (cycle 2 et 8).

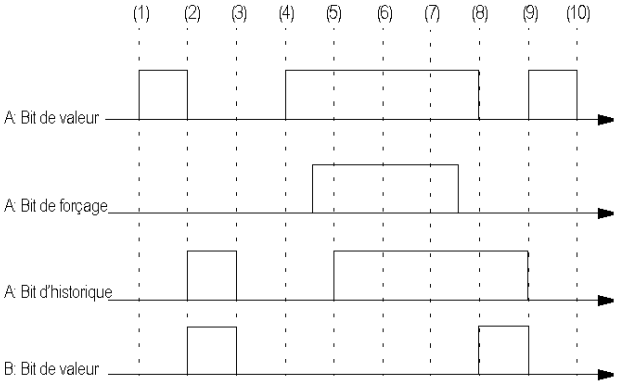


Forçage de bits

Lors du forçage de bits, la valeur des variables indiquée par la logique est écrasée par la valeur de forçage.

Dans l'exemple, un front négatif de la variable A doit être reconnu et B doit donc être défini pendant un cycle.

Chaque fois que le bit de valeur ou le bit de forçage de A est égal à 0 et que le bit d'historique est égal à 1, B est réglé sur 1 pendant un cycle (cycle 1 et 8).



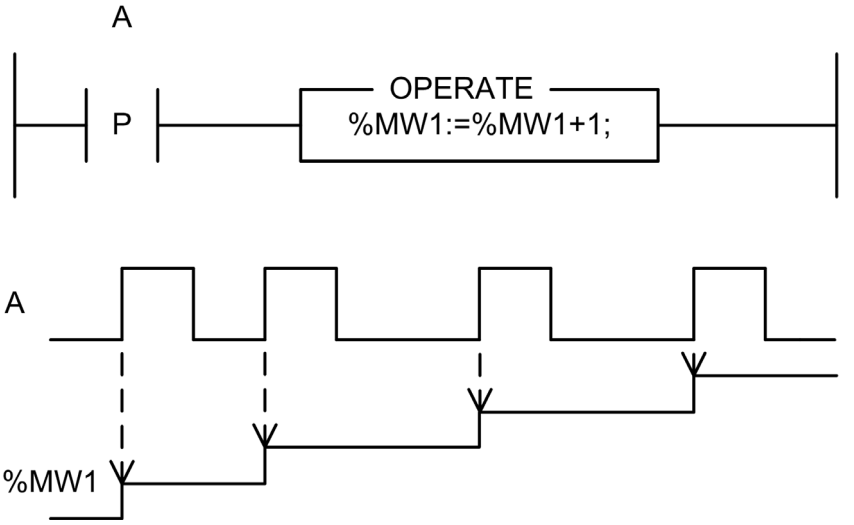
Utilisation des variables BOOL et EBOOL

Le comportement de la détection de fronts peut varier selon que vous utilisez le type de variable `BOOL` ou `EBOOL` :

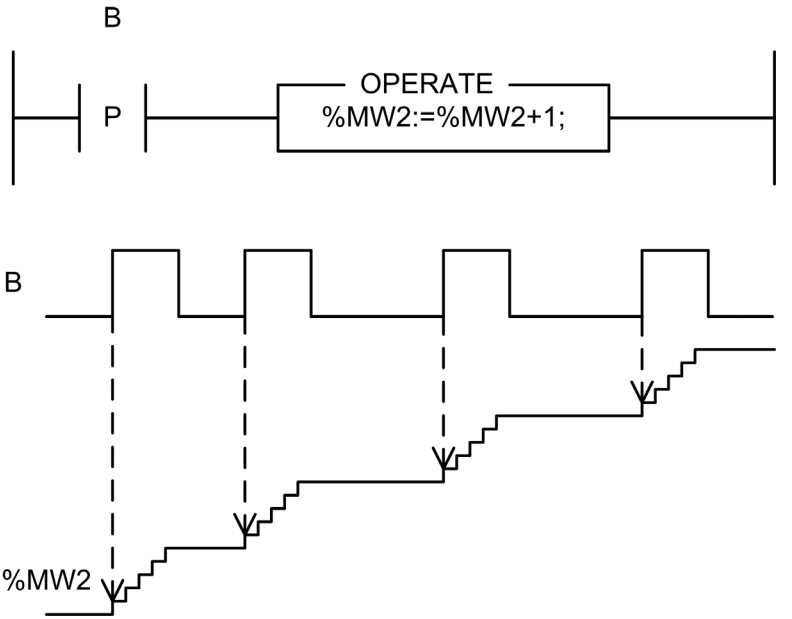
- Lorsque vous utilisez une variable `BOOL`, le système gère l'historique en permettant la détection de front pendant l'exécution du contact.
- Lorsque vous utilisez une variable `EBOOL`, le bit d'historique est actualisé pendant l'exécution de la bobine.

Les exemples suivants montrent les différents comportements selon le type de variable.

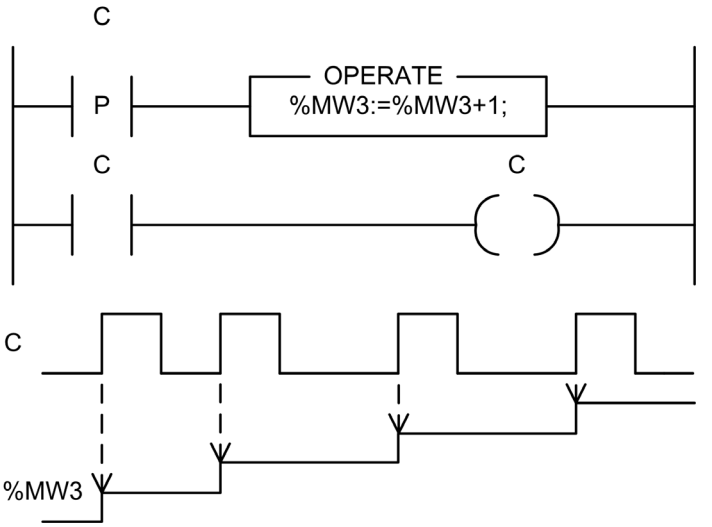
La variable `A` est définie comme `BOOL` ; chaque fois que `A` est réglé sur 1, `%MW1` est incrémenté de 1.



La variable B est définie comme EBOOL ; son comportement diffère de celui de la variable A. Lorsque B est réglé sur 1, %MW2 est incrémenté de 1 parce que le bit d'historique n'est pas actualisé.



La variable C est définie comme EBOOL ; son comportement est identique à celui de la variable A. Le bit d'historique est actualisé.

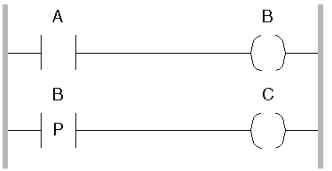


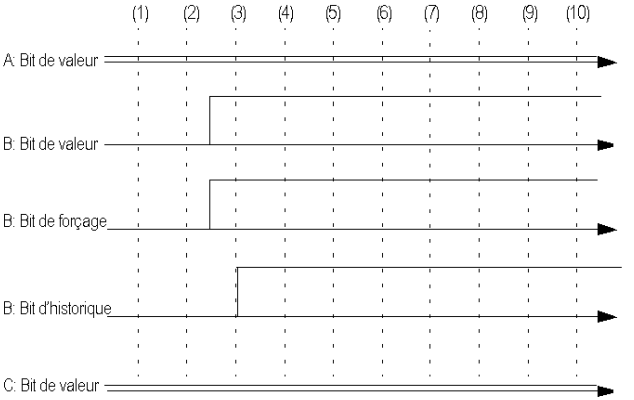
Perte de la détection de fronts probablement due au forçage de bobines

Le forçage de bobines peut causer la perte de la détection de fronts.

Dans l'exemple, lorsque A est égal à 1, B devrait être égal à 1, et avec un front montant de A, la bobine B sera définie pendant un cycle.

Dans cet exemple, la variable B est d'abord affectée à la bobine, puis à la liaison pour reconnaître les fronts montants.





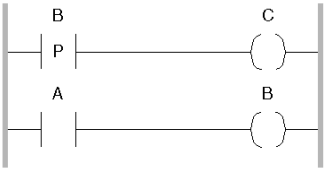
Au début du deuxième cycle, le bit de valeur de B est égal à 0. En cas de forçage de B dans ce cycle, le bit de forçage et le bit de valeur sont réglés sur 1. Pendant le traitement de la première ligne de logique dans le troisième cycle, le bit d'historique de la bobine (B) est également réglé sur 1.

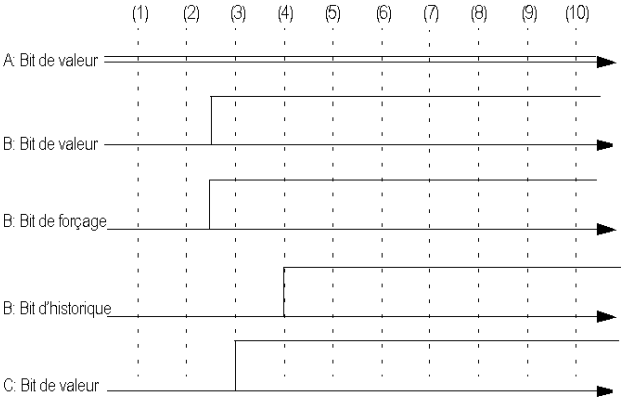
Problème :

Pendant la détection de front (comparaison du bit de valeur et du bit d'historique), dans la deuxième ligne de logique, aucun front n'est détecté car, en raison de la lise à jour, le bit de valeur et d'historique de la ligne 1 de B sont toujours identiques.

Solution :

Dans cet exemple, la variable B est d'abord affectée à la liaison pour reconnaître les fronts montants, puis à la bobine.





Au début du deuxième cycle, le bit de valeur de B est égal à 0. En cas de forçage de B dans ce cycle, le bit de forçage et le bit de valeur sont réglés sur 1. Pendant le traitement de la première ligne de logique dans le troisième cycle, le bit d'historique de la liaison (B) reste réglé sur 0.

La détection de front reconnaît la différence entre les bits de valeur et les bits d'historique et règle la bobine (C) sur 1 pour un cycle.

Perte de la détection de fronts probablement due à l'utilisation des bobines d'enclenchement ou de déclenchement

L'utilisation de bobines d'enclenchement ou de déclenchement peut causer la perte de la détection de fronts avec les variables EBOOL.

La variable au-dessus de la bobine d'enclenchement/de déclenchement (variable C dans l'exemple) est toujours affectée par la valeur de la liaison de gauche.

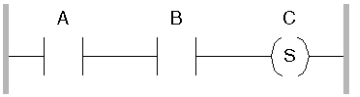
Si la liaison de gauche est 1, le bit de valeur bit (variable C dans l'exemple) est copié dans le bit d'historique et le bit de valeur est réglé sur 1.

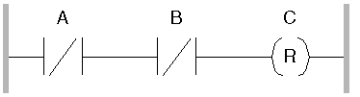
Si la liaison de gauche est 0, le bit de valeur bit (variable C dans l'exemple) est copié dans le bit d'historique, mais le bit de valeur n'est pas modifié.

Ainsi, quelle que soit la valeur prise par la liaison de gauche avant la bobine d'enclenchement ou de déclenchement, le bit d'historique est toujours mis à jour.

Dans l'exemple, un front positif de la variable C devrait être reconnu et définir D pendant un cycle.

Ligne de code	Comportement en LD	Equivalence en ST

1	Situation initiale : C = **0**, Bit d'historique = **0**
	 
	A = **1**,
	B = **1**,
	C = **1**, Bit d'historique = **0**
		`IF A AND B`
		`  THEN C := 1;`
		`ELSE C := C;`
		`END_IF;`

2	
	A = **1**,
	B = **1**,
	C = **1**, Bit d'historique = **1**
		`IF NOT(A) AND NOT(B)`
		`THEN  C := 0;`
		`ELSE C := C;`
		`END_IF;`

3	 C = **1**, Bit d'historique = **1** D = **0**, car le bit de valeur et le bit d'historique de C sont identiques. Le front montant de C, représenté dans ligne 1 du code, n'est par reconnu par le code de la ligne 2, car cela force l'actualisation du bit d'historique. (Si la condition est FALSE, la valeur présente de C est à nouveau attribuée à C, voir l'instruction ELSE de la ligne de code 2 dans l'exemple ST, par exemple.)	-

Ordre d'exécution et flux de signaux

Ordre d'exécution des réseaux

Les règles suivantes s'appliquent à l'ordre d'exécution des réseaux :

- l'exécution d'une section a lieu réseau pour réseau via les liaisons d'objets du haut vers le bas.
- les boucles ne peuvent pas être configurées par le biais de liaisons, étant donné que, dans ce cas, l'ordre d'exécution ne peut pas être défini de façon unique. Les boucles doivent être résolues par le biais de paramètres réels (voir *Configuration de boucles, page 392*).
- l'ordre d'exécution de réseaux reliés ensemble uniquement par la barre gauche d'alimentation est déterminé par l'ordre graphique (du haut vers le bas) dans lequel ces réseaux sont reliés à la barre gauche d'alimentation. Cela ne s'applique pas si l'ordre a été influencé par des éléments de commande.
- le calcul d'un réseau doit être complètement terminé avant que le calcul du réseau suivant puisse commencer.
- aucun élément d'un réseau n'est considéré comme calculé avant que l'état de toutes les entrées de cet élément n'ait été calculé.
- le calcul d'un réseau est considéré comme terminé lorsque toutes les sorties de ce réseau sont calculées. Cela s'applique également si le réseau comprend un ou plusieurs éléments de commande.

Flux de signaux dans un réseau

Les règles suivantes s'appliquent au flux de signaux au sein d'un réseau (rung) :

- le flux de signaux pour les liaisons booléennes est
  - de la gauche vers la droite pour les liaisons booléennes horizontales et
  - du haut vers le bas pour les liaisons booléennes verticales.
- le flux de signaux d'une liaison FFB va de la sortie FFB vers l'entrée FFB, indépendamment de la direction.
- un FFB n'est calculé que lorsque tous les éléments (sorties FFB, etc.) qui sont reliés à ses entrées sont calculés.
- l'ordre d'exécution des FFB reliés à différentes sorties du même FFB va du haut vers le bas.
- l'ordre d'exécution des objets n'est pas influencé par leur position au sein du réseau.
- l'ordre d'exécution de FFB apparaît sous forme de numéro d'exécution au dessus du FFB.

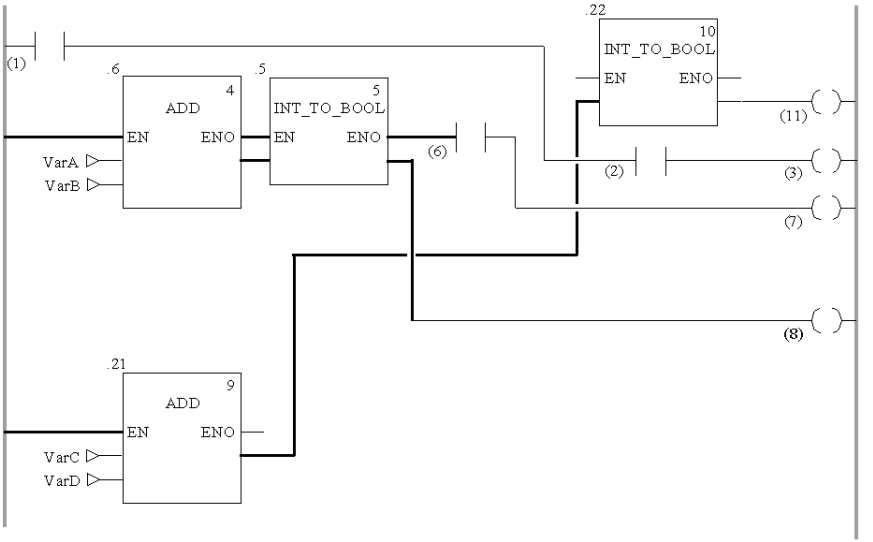
Priorités

Priorités lors de la détermination du flux de signaux au sein d'une section :

Priorité	Règle	Description
1	Liaison	Les liaisons ont la priorité la plus élevée lors de la détermination du flux de signaux au sein d'une section LD.
2	Réseau pour réseau	Le calcul d'un réseau doit être complètement terminé avant que le calcul du réseau suivant puisse commencer.
3	Ordre des sorties	Les sorties du même bloc fonction ou les sorties de liaisons verticales sont calculées du haut vers le bas.
4	Rung pour rung	Priorité la moins élevée. L'ordre d'exécution des réseaux reliés ensemble uniquement par la barre gauche d'alimentation est déterminé par l'ordre graphique (du haut vers le bas) dans lequel ces réseaux sont reliés à la barre gauche d'alimentation. (Cela ne s'applique que si aucune autre règle n'intervient.)

Exemple

Exemple de l'ordre d'exécution des objets dans une section LD :



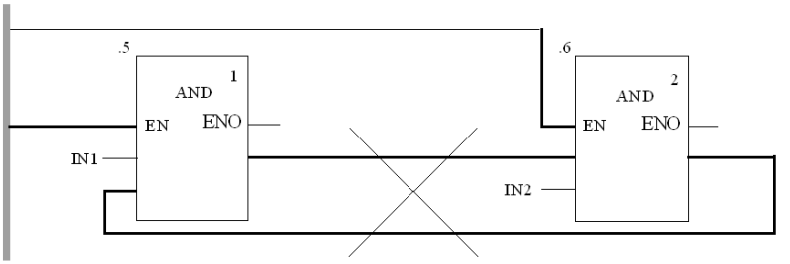
NOTE : Les numéros d'exécution de contacts et de bobines n'est pas affiché. Ils n'ont été indiqués dans le graphique que pour fournir une meilleure vue d'ensemble.

Configuration de boucles

Boucles non permises

La configuration de boucles exclusivement par le biais de liaisons n'est pas permise, étant donné que, dans ce cas, une détermination unique du flux de signaux n'est pas possible (la sortie d'un FFB est l'entrée du FFB suivant, et la sortie de celui-ci est à son tour l'entrée du premier).

Boucles non permises par le biais de liaisons :



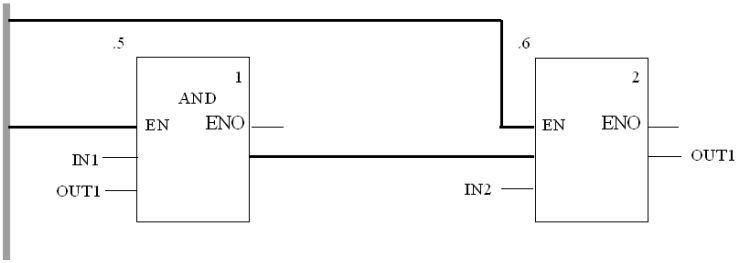
Résolution par le biais d'un paramètre réel

Une telle logique doit être résolue par le biais de variables de réaction, afin que le flux de signaux puisse être défini de façon unique.

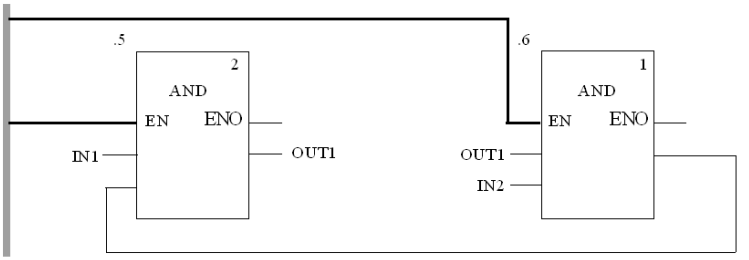
Les variables de réaction doivent être initialisées. La valeur initiale est utilisée lors de la première exécution de la logique. Une fois la première exécution effectuée, la valeur initiale est remplacée par la valeur actuelle.

Respectez pour les deux variantes l'ordre d'exécution (numéro entre parenthèses après le nom d'instance) des deux blocs.

Boucle résolue par le biais d'un paramètre réel : Variante 1



Boucle résolue par le biais d'un paramètre réel : Variante 2



Modification de l'ordre d'exécution

Présentation

L'ordre d'exécution des réseaux et l'ordre d'exécution des objets au sein d'un réseau sont définis par une série de règles *(voir page 390)*.

Dans certains cas, il est nécessaire de modifier l'ordre d'exécution proposé par le système.

Pour définir/modifier l'ordre d'exécution des réseaux, vous disposez des possibilités suivantes :

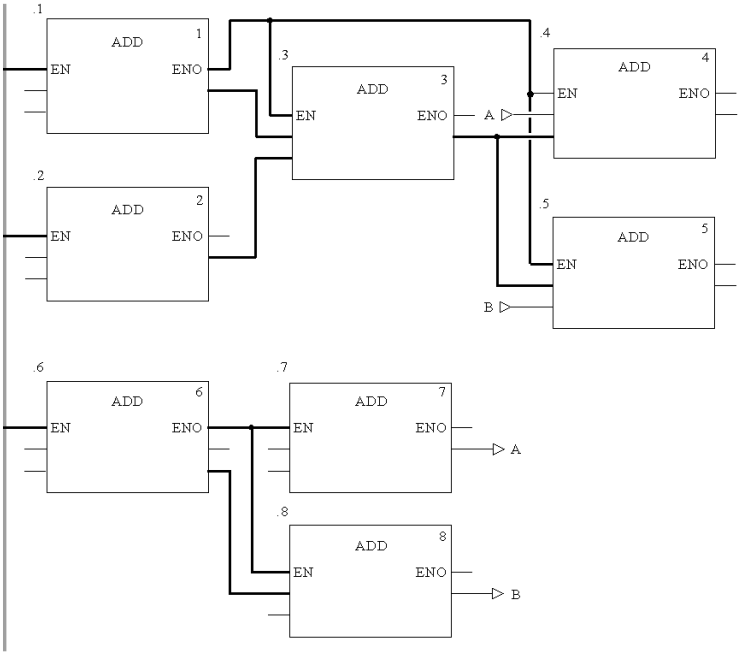
- utiliser des liaisons au lieu des paramètres réels,
- position des réseaux,

Pour définir/modifier l'ordre d'exécution au sein des réseaux, vous disposez des possibilités suivantes :

- position des objets.

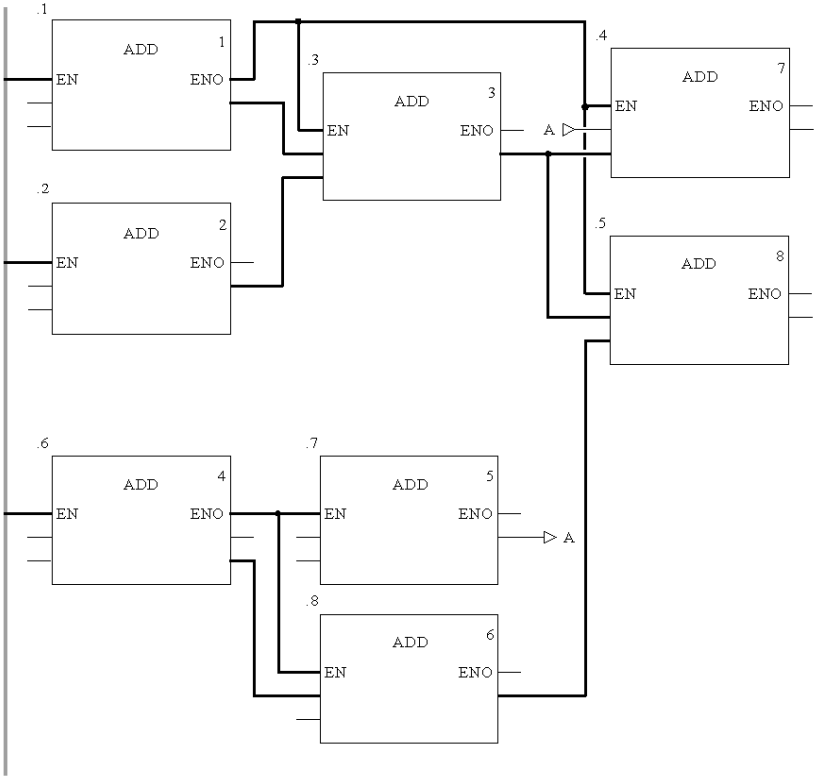
Situation de sortie

L'image suivante illustre deux réseaux dont l'ordre d'exécution est déterminé uniquement par leur position au sein de la section, indépendamment du fait que les blocs `0,4/0,5` et `0,7/0,8` nécessitent un ordre d'exécution différent.



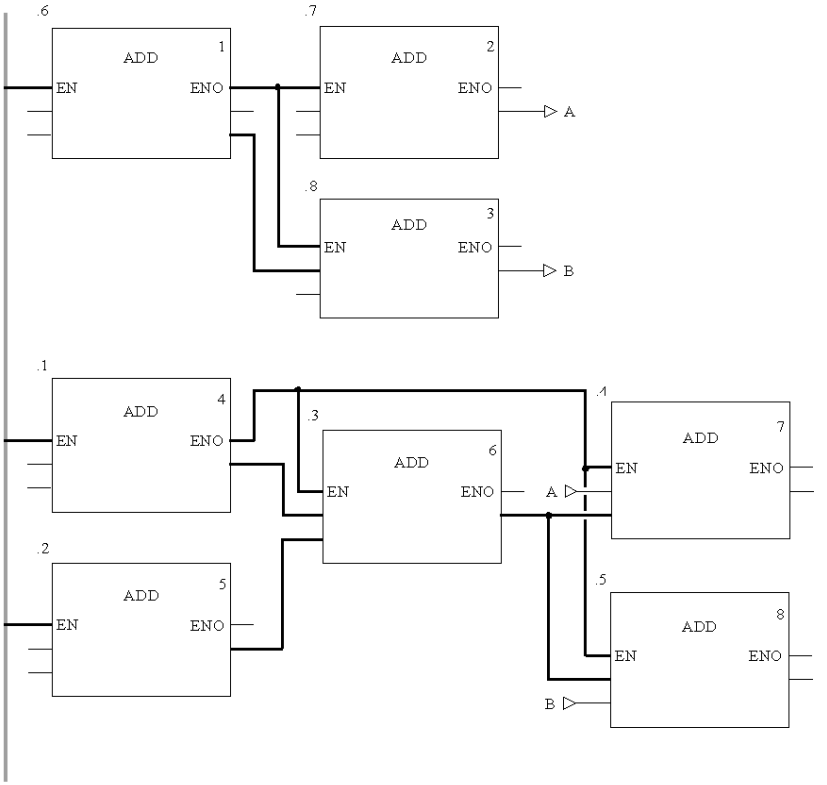
Liaison au lieu d'un paramètre réel

Lorsque l'on utilise une liaison au lieu d'une variable, les deux réseaux sont exécutés dans l'ordre correct (voir également *Situation de sortie, page 394*).



Position des réseaux

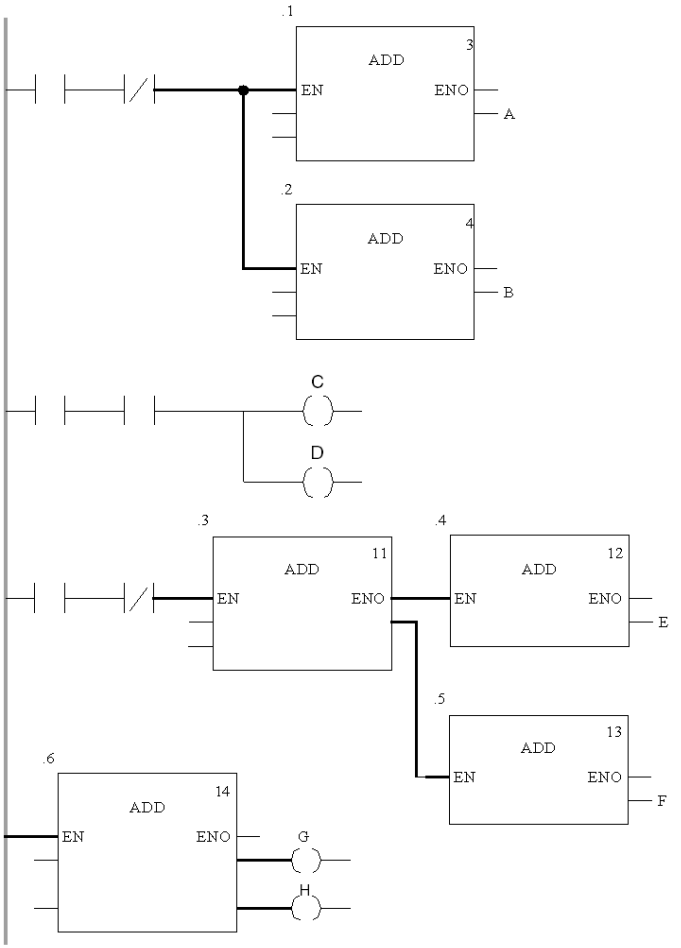
L'ordre d'exécution correct peut être obtenu en modifiant les positions des réseaux dans la section (voir également *Situation de sortie, page 394*).



Position des objets

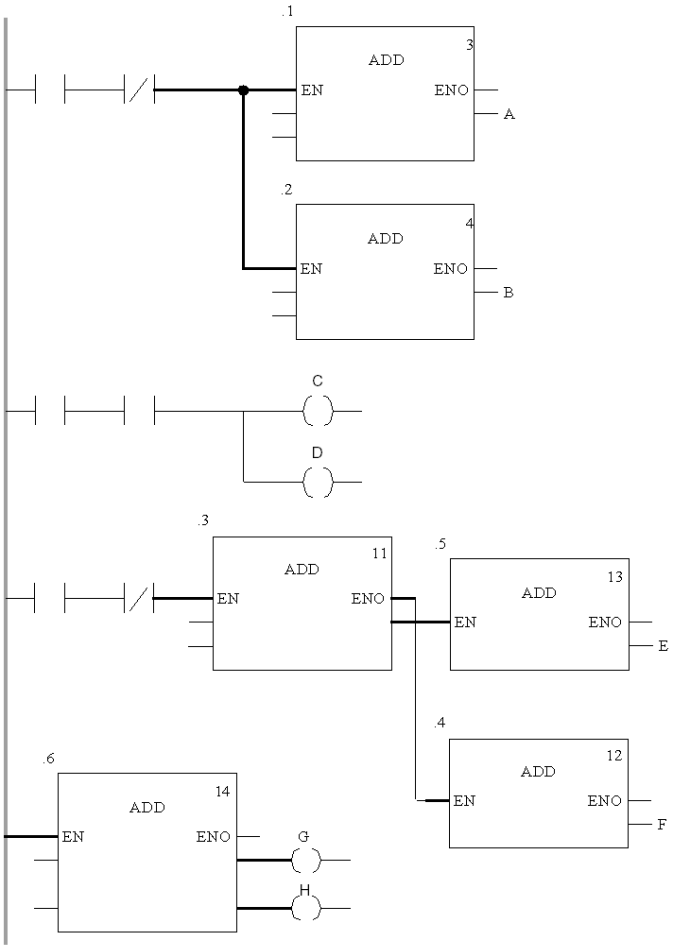
La position des objets influe alors sur l'ordre d'exécution uniquement si plusieurs entrées (liaison gauche de contacts/bobines, entrées FFB) sont reliées à la même sortie de l'objet "appelant" (liaison droite de contacts/bobines, sorties FFB) (voir également *Situation de sortie, page 394*).

Situation de sortie :



Les positions du bloc `0,1` et `0,2` sont permutées dans le premier réseau. Dans ce cas (source commune des deux entrées de bloc), l'ordre d'exécution des deux blocs est également permuté (traitement du haut vers le bas). La même chose s'applique pour la permutation des bobines `C` et `D` dans le deuxième réseau.

Les positions du bloc 0,4 et 0,5 sont permutées dans le troisième réseau. Dans ce cas (source différente des entrées de bloc) l'ordre d'exécution des deux blocs n'est pas permuté (traitement dans l'ordre des sorties de bloc à appeler). La même chose s'applique pour la permutation des bobines G et H dans le dernier réseau.



Langage séquentiel SFC

13

Objet de ce chapitre

Ce chapitre décrit le langage séquentiel SFC conforme à la norme CEI 611311.

Contenu de ce chapitre

Ce chapitre contient les sous-chapitres suivants :

13.1 Généralités concernant le diagramme fonctionnel en séquence (SFC)

Objet de ce sous-chapitre

Ce sous-chapitre vous donne un aperçu général du diagramme fonctionnel en séquence SFC.

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Informations générales sur le diagramme fonctionnel en séquence SFC

Présentation

Cette section décrit le langage séquentiel SFC (Diagramme fonctionnel en séquence), conforme à la norme CEI 61131-3.

Structure d'un diagramme fonctionnel en séquence

Un diagramme fonctionnel en séquence conforme à CEI se compose dans Unity Pro de sections SFC (niveau supérieur), de sections de transition et de sections d'action.

Ces sections SFC ne sont admises que dans la tâche maître du projet. Dans d'autres tâches ou DFB, les sections SFC ne peuvent pas être utilisées.

Chaque section SFC contient exactement un réseau SFC (séquence) dans le jeton unique.

Les jetons multiples d'une section SFC peuvent contenir un ou plusieurs réseaux SFC indépendants les uns des autres.

Objets

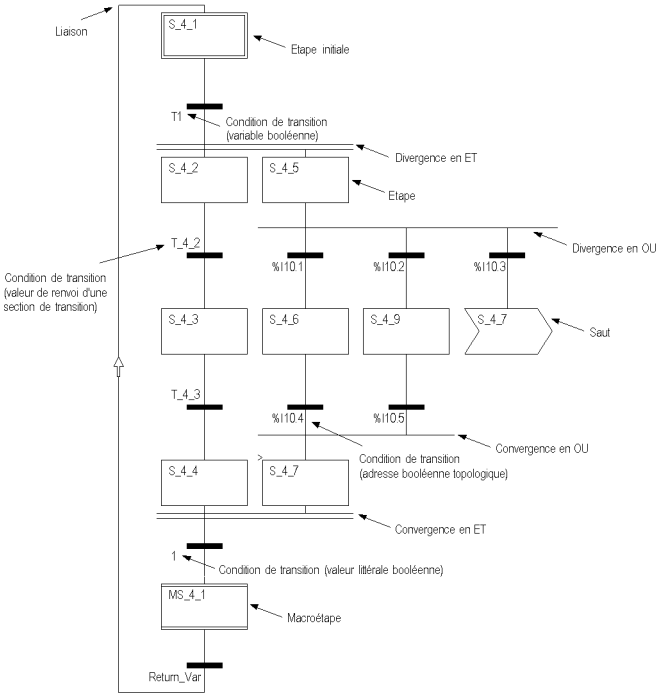
Une section SFC contient les objets de création de programme suivants :

- étape *(voir page 406)*
- macroétape (séquence de sous-étapes imbriquées) *(voir page 411)*
- transition (condition de transition) *(voir page 423)*
- saut *(voir page 427)*
- liaison *(voir page 428)*
- divergence en OU *(voir page 430)*
- convergence en OU *(voir page 430)*
- divergence en ET *(voir page 431)*
- convergence en ET *(voir page 431)*

La logique de la section peut être commentée par des objets texte (voir *Objet texte, page 432*).

Représentation d'une section SFC

Représentation :



Structure d'une section SFC

Une section SFC est une "machine d'états", c.-à-d. que l'état est déterminé par les étapes actives et les transitions renvoient le comportement de commutation/modification entre les états. Les étapes et transitions sont reliées les unes aux autres par des liaisons dirigées. Deux étapes consécutives ne peuvent jamais être directement reliées ; elles sont toujours séparées par une transition. Les évolutions des états actifs de signaux se déroulent le long des liaisons dirigées, et sont déclenchées par la commutation d'une transition. Le déroulement d'une séquence va dans le sens des liaisons dirigées et se déroule de la partie inférieure de l'étape précédente à la partie supérieure de l'étape suivante. Les divergences sont traitées de gauche à droite.

Chaque étape peut compter zéro ou plusieurs actions. A chaque transition est associée une condition de transition.

La dernière transition de la séquence est toujours reliée à une autre étape de la séquence (par une liaison graphique ou un symbole de "saut") de manière à obtenir une boucle fermée. Les séquences d'étapes se déroulent donc de façon cyclique.

SFCCHART_STATE Variable

A la création d'une section SFC, une variable du type de données SFCCHART_STATE lui est automatiquement affectée. La variable ainsi créée porte toujours le nom de la section SFC correspondante.

Cette variable sert à affecter les blocs de commande SFC à la section SFC à commander.

Règle de jetons

Le comportement d'un réseau SFC dépend largement du nombre de jetons choisis, c.-à-d. du nombre d'étapes actives.

Un comportement univoque est possible en utilisant un seul jeton (single token). Les divergences en ET comportant un jeton actif (étape) par branche sont considérées comme des jetons uniques. Ceci correspond à une séquence d'étapes selon la norme CEI 61131-3.

Une séquence d'étapes comportant un maximum d'étapes actives (jetons multiples) définies par l'utilisateur augmente le niveau de liberté. Les limitations relatives à l'obligation d'unicité et du non-blocage sont à cet effet levées et doivent être assurées par l'utilisateur. Les séquences d'étape à jetons multiples ne sont pas conformes à la norme CEI 61131-3.

Taille de la section

- Une section SFC se compose d'une fenêtre comportant une seule page.
- Pour des raisons de performance, il est recommandé de créer moins de 100 sections SFC dans un projet (les sections macro ne sont pas comptabilisées).
- Cette fenêtre comporte une grille logique de 200 lignes et de 32 colonnes.
- Les étapes, les transitions et les sauts requièrent tous une cellule.
- Les divergences et convergences ne nécessitent pas de cellule propre, mais sont insérées dans la cellule correspondante de l'étape ou de la transition.
- Chaque section SFC (avec toutes ses macrosections) peut contenir jusqu'à 1024 étapes.
- Il est possible d'activer jusqu'à 100 étapes (jetons multiples) par section SFC (avec toutes ses macrosections).
- Il est possible de placer manuellement jusqu'à 64 étapes simultanément par section SFC (jetons multiples).
- 20 actions, au maximum, peuvent être affectées à chaque étape SFC.
- La possibilité d'imbrication des macros, c'est-à-dire "macroétape dans macroétape", s'élève à huit niveaux.

Conformité CEI

Pour la description de la conformité CEI du langage SFC, voir Conformité CEI *(voir page 657)*.

Règles de liaison

Règles de liaison

Ce tableau indique les connexions que vous pouvez établir entre les sorties d'objet et les entrées d'objet.

Depuis la sortie de l'objet	Vers l'entrée de l'objet
Etape	Transition
	Divergence en OU
	Convergence en ET
Transition	Etape
	Saut
	Divergence en ET
	Convergence en OU
Divergence en OU	Transition
Convergence en OU	Etape
	Saut
	Divergence en ET
	Convergence en OU
Divergence en ET	Etape
	Saut
	Convergence en OU (uniquement pour les jetons multiples *(voir page 444)*)
Convergence en ET	Transition
	Divergence en OU (uniquement pour jetons multiples *(voir page 444)*)
	Convergence en OU

13.2 Etape et macro-étape

Objet de ce sous-chapitre

Ce sous-chapitre décrit les objets étape et macro-étape du diagramme fonctionnel en séquence (SFC).

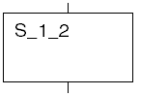
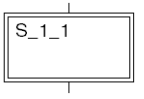
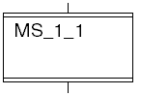
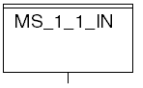
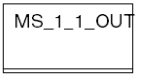
Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Etape

Types d'étape

Les étapes se composent des types suivants :

Type	Représentation	Description
Etape "normale"		Une étape devient active lorsque l'étape précédente devient inactive (un temps de retard éventuellement défini doit s'être écoulé) et la transition située en amont est vraie. Une étape devient généralement inactive lorsque le temps de retard éventuellement défini s'est écoulé et que la transition située en aval est vraie. Pour les convergences en ET, toutes les étapes précédentes doivent être vraies Chaque étape compte zéro ou plusieurs actions. Les étapes sans action sont considérées comme des étapes d'attente.
Etape initiale		L'état initial d'une séquence est caractérisé par l'étape initiale. A l'issue de l'initialisation du projet ou de la séquence, l'étape initiale est active. Généralement, aucune action n'est affectée aux étapes initiales. Pour les jetons uniques (conformes à la norme CEI 61131-3), seule une étape initiale est admise par séquence. Pour les jetons multiples un nombre d'étapes initiales pouvant être défini (de 0 à 100) est possible.
Macroétape		voir *Macroétape, page 411*
Etape d'entrée		voir *Etape d'entrée, page 411*
Etape de sortie		voir *Etape de sortie, page 412*

Noms d'étape

A chaque création d'étape, un numéro de proposition lui est affecté. La structure du numéro proposé est la suivante : `S_i_j`, où `i` représente le numéro reel (interne) de la section et `j` est le numéro de l'étape actuelle (interne) dans la section.

Vous pouvez changer ces numéros de proposition pour avoir une meilleure vue d'ensemble. Les noms d'étape (32 caractères maximum) doivent être uniques dans l'ensemble du projet, c-à-d. qu'il ne doit pas exister d'autres étapes, variables, sections etc., ayant le même nom. Aucune distinction n'est faite entre l'écriture en majuscules et en minuscules. Le nom d'étape doit répondre aux conventions sur les noms.

Temps d'étape

Un temps minimum et maximum de contrôle ainsi qu'un temps de retard peuvent être affectés à chaque étape.

- **Temps de contrôle minimum**
  Le temps minimum de contrôle indique la durée minimale pendant laquelle l'étape doit être active. Si l'étape devient inactive avant que ce délai ne soit écoulé, un message d'erreur s'affiche. En mode d'animation, l'erreur est de plus signalée par un changement de couleur (jaune) de l'objet d'étape.
  Si vous n'indiquez pas de temps minimum de contrôle ou que vous indiquez un temps minimum nul, un contrôle de l'étape n'est pas exécuté.
  L'état d'erreur est conservé jusqu'à ce que l'étape redevienne active.
- **Temps maximum de contrôle**
  Le temps maximum de contrôle indique la durée maximale pendant laquelle l'étape doit être normalement active. Si l'étape est toujours active lorsque ce délai est écoulé, un message d'erreur apparaît. En mode d'animation, l'erreur est de plus signalée par un changement de couleur (rose) de l'objet d'étape.
  Si vous n'indiquez pas de temps maximum de contrôle ou que vous indiquez un temps maximum nul, un contrôle de l'étape n'est pas exécuté.
  L'état d'erreur est conservé jusqu'a ce que l'étape devienne inactive.
- **Temps de retard**
  Le temps de retard (palier de l'étape) indique la durée minimale pendant laquelle l'étape doit être active.

NOTE : Les durées indiquées ne sont valables que pour l'étape, pas pour les actions qui y sont associées. Pour celles-ci, il est possible de définir des temps propres.

Définition des temps d'étapes

Lors de la définition/détermination des temps, veuillez tenir compte de la formule suivante :

```
Temps de retard < temps minimum de contrôle < temps maximum de contrôle
```

On distingue 2 possibilités d'affectation des valeurs définies/déterminées à une étape :

- saisie sous forme de libellé de durée
- utilisation de la structure de données `SFCSTEP_TIMES`

Variable SFCSTEP_TIMES

Vous pouvez affecter à chaque étape une variable du type de données SFCSTEP_TIMES. Les éléments de cette structure de données sont accessibles en lecture et en écriture (read/write).

La structure de données est traitée comme toute autre structure de données, c.-à-d. qu'elle peut être utilisée dans des déclarations de variables et il est donc possible d'y accéder dans son ensemble (p. ex. en tant que paramètre FFB).

Description de la structure de données :

Nom de l'élément	Type	Description
"VarName".delay	TIME	Temps de retard
"VarName".min	TIME	Temps de contrôle minimum
"VarName".max	TIME	Temps maximum de contrôle

Variable SFCSTEP_STATE

A chaque étape est affectée de façon implicite une variable du type de données SFCSTEP_STATE. Cette variable d'étape porte le nom de l'étape à laquelle elle est affectée. Les éléments de cette structure de données ne sont accessibles qu'en lecture (read-only).

Vous pouvez voir les variables SFCSTEP_STATE dans l' **Editeur de données**. Le **Commentaire** pour une variable SFCSTEP_STATE est le commentaire entré comme propriété de l'étape elle-même. Consultez la section "Définition des propriétés d'étapes" *(voir Unity Pro, Modes de marche,)* dans le *manuel des modes de fonctionnement Unity Pro*.

La structure de données ne peut pas être utilisée dans des déclarations de variables. C'est pourquoi il est impossible d'accéder à la structure de données dans son ensemble (p. ex. sous forme de paramètre FFB).

Description de la structure de données :

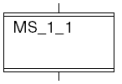
Nom de l'élément	Type	Description
"Nom d'étape" t	TIME	Temps d'activation actuel dans l'étape. Lorsque l'étape est désactivée, la valeur de cet élément est maintenue jusqu'à la prochaine activation de l'étape.
"Nom d'étape" .x	BOOL	1: Etape active 0: Etape inactive
"Nom d'étape".tminErr	BOOL	Cet élément est une extension de la norme CEI 61131-3. 1: dépassement par valeur inférieure du temps minimum de contrôle 0: pas de dépassement par valeur inférieure du temps minimum de contrôle Dans les cas suivants, l'élément est automatiquement réinitialisé : ● lorsque l'étape est réactivée, ● lorsque la commande de séquence est réinitialisée, ● Si le bouton de commande **RAZ erreur durée** est activé
"Nom d'étape".tmaxErr	BOOL	Cet élément est une extension de la norme CEI 61131-3. 1: Dépassement du temps de contrôle maximum 0: Pas de dépassement du temps maximum de contrôle Dans les cas suivants, l'élément est automatiquement réinitialisé : ● à la sortie de l'étape, ● lorsque la commande de séquence est réinitialisée, ● Si le bouton de commande **RAZ erreur durée** est activé

Macro-étapes et macro-sections

Macroétape

Les macro-étapes servent à appeler une macro-section et ainsi à établir une structure hiérarchique des commandes d'enchaînement.

Représentation d'une macroétape :



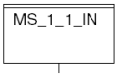
Les macroétapes ont les caractéristiques suivantes :
- elles se positionnent en sections de "commande de séquence" et en macro-sections,
- le nombre de macroétapes n'est pas limité,
- le degré d'imbrication des macroétapes l'une dans l'autre est de 8 niveaux,
- à chaque macroétape est affectée de manière implicite une variable du type de données `SFCSTEP_STATE`, voir *Variable SFCSTEP_STATE, page 409*,
- à chaque macroétape peut être affectée une variable du type de données `SFCSTEP_TIMES`, voir *Variable SFCSTEP_TIMES, page 409*,
- AUCUNE action ne peut être affectée aux macroétapes,
- Chaque macroétape peut être remplacée par la séquence contenue dans la macro-section correspondante.

Les macroétapes sont une extension de la norme CEI 61131-3 et doivent être activées de manière explicite.

Etape d'entrée

Chaque macro-section commence par une étape d'entrée.

Représentation d'une étape d'entrée :

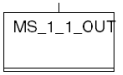


Les étapes d'entrée ont les caractéristiques suivantes :
- les étapes d'entrée sont automatiquement placées dans des sections de macro par l'éditeur SFC,
- seule une étape d'entrée est placée par macro-section,
- une étape d'entrée ne peut pas être supprimée, copiée ni insérée manuellement,
- à chaque étape d'entrée est affectée de manière implicite une variable du type de données `SFCSTEP_STATE`, voir *Variable SFCSTEP_STATE, page 409*,
- à chaque étape d'entrée peut être affectée une variable du type de données `SFCSTEP_TIMES`, voir *Variable SFCSTEP_TIMES, page 409*,
- des actions peuvent être affectées à des étapes d'entrée.

Etape de sortie

Chaque macro-section se termine par une étape de sortie.

Représentation d'une étape de sortie :



Une étape de sortie a les caractéristiques suivantes :
- les étapes de sortie sont automatiquement placées dans des sections de macro par l'éditeur SFC,
- seule une étape de sortie est placée par macro-section,
- une étape de sortie ne peut pas être supprimée, copiée ni insérée manuellement,
- AUCUNE action ne peut être affectée aux étapes de sortie,
- les étapes de sortie ne peuvent se voir affecter qu'un temps de retard, l'affectation de temps de contrôle n'est pas possible, voir *Temps d'étape, page 407*.

Macro-section

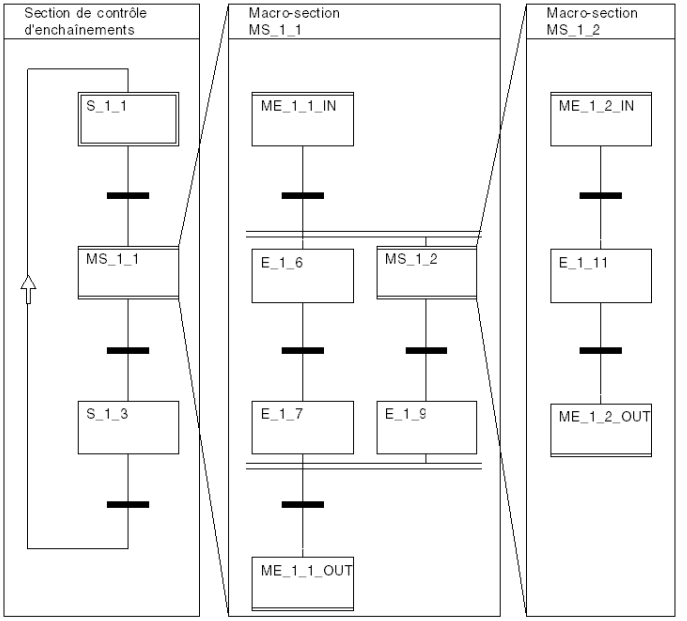
Une macro-section se compose d'une séquence unique, laquelle dispose en principe des mêmes éléments que les sections de "commande de séquence" (p.ex. étapes, étape(s) initiale(s), macro-étapes, transitions, divergences, convergences etc.).

De plus, chaque macro-section comporte en début une étape d'entrée et en fin une étape de sortie.

Chaque macroétape peut être remplacée par la séquence contenue dans la macro-section correspondante.

Ainsi les macro-sections peuvent comporter 0, 1 ou plusieurs étapes initiales, voir également *Types d'étape, page 406*.
- Jeton unique
  - 0 étape(s) initiale(s)
    sont utilisées dans les macro-sections si une étape initiale est déjà disponible dans la section de rang hiérarchique supérieur ou inférieur.
  - 1 étape(s) initiale(s)
    sont utilisées dans les macro-sections si aucune étape initiale n'est disponible dans la section de rang hiérarchique supérieur ou inférieur.
- Jetons multiples
  Un maximum de 100 étapes initiales peuvent être utilisées dans chaque section (y compris toutes leurs macro-sections).

Utilisation de macro-sections :



Le nom des macro-sections est identique au nom de la macroétape appelante. Si le nom de la macroétape est modifié, le nom de la macro-section correspondante l'est aussi automatiquement.

Une macro-section ne peut être utilisée qu'une seule fois.

Exécution de macroétapes

Exécution des macroétapes :

Phase	Description
1	Une macroétape devient active lorsque la condition de transition qui la précède est VRAIE. L'étape d'entrée de la macro-section devient également active.
2	La séquence de la macro-section est exécutée. La macro-étape reste active tant qu'au moins une étape de la macro-section est active.
3	Si l'étape de sortie de la macro-section est active, la transition suivant la macroétape est validée.
4	La macroétape devient inactive lorsque l'étape de sortie est active et donc que la condition de transition suivante est validée et que la condition de transition est VRAIE. L'étape de sortie de la macro-section devient également inactive.

Noms d'étape

A chaque création d'étape, un numéro de proposition lui est affecté.

Signification des numéros de proposition :

Type d'étape	Numéro de proposition	Description
Macroétape	`ME_i_j`	`ME` = Macroétape `i` = numéro courant (interne) de la section actuelle `j` = numéro de macro-étape courant (interne) de la section actuelle
Etape d'entrée	`ME_k_l_IN`	`ME` = Macroétape `k` = numéro courant (interne) de la section qui appelle `l` = numéro de macro-étape courant (interne) dans la section qui appelle `IN` = étape d'entrée
Etape de sortie	`ME_k_l_OUT`	`ME` = Macroétape `k` = numéro courant (interne) de la section qui appelle `l` = numéro de macro-étape courant (interne) dans la section qui appelle `OUT` = étape de sortie
Etape "normale" (dans une macro-section)	`E_k_m`	`E` = étape `k` = numéro courant (interne) de la section qui appelle `m` = numéro d'étape courant (interne) de la section appelante

Vous pouvez changer ces numéros de proposition pour avoir une meilleure vue d'ensemble. Les noms d'étape (28 caractères maximum pour les noms de macroétapes, 32 caractères maximum pour les noms d'étapes) doivent être uniques dans l'ensemble du projet, c.-à-d. qu'il ne doit pas exister d'autres étapes, variables, sections (excepté le nom de la macro-section affectée à la macroétape) etc., ayant le même nom. Aucune distinction n'est faite entre l'écriture en majuscules et en minuscules. Le nom d'étape doit répondre aux conventions sur les noms.

Si le nom de la macroétape est modifié, le nom de la macro-section correspondante et des étapes qu'elle contient le sont aussi automatiquement..

Exemple : si `ME_1_1` est renommé en MyStep, les noms d'étape de la macro-section sont transformés en `MyStep_IN`, `MyStep_1`, ..., `MyStep_n`, `MyStep_OUT`.

13.3 Action et section d'action

Objet de ce sous-chapitre

Ce sous-chapitre décrit les actions et les sections d'action du du diagramme fonctionnel en séquence SFC

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Action

Présentation

Les actions ont les caractéristiques suivantes :

- une action peut être une variable booléenne (variable d'action *(voir page 417)*) ou une section (section d'action *(voir page 418)*) du langage de programmation FBD, LD, IL ou ST.
- il est possible d'affecter aucune ou plusieurs actions à une étape. Une étape à laquelle aucune action n'est affectée a une fonction d'attente, c.-à-d. qu'elle attend que la transition affectée soit vérifiée.
- si plusieurs actions sont affectées à une même étape, elles sont traitées dans l'ordre de leur occurrence dans la zone de liste action.
  Exception : Indépendamment de leur position dans la zone de liste action, les actions comportant l'identificateur *(voir page 419)* `P1` sont toujours traitées en premier et les actions comportant l'identificateur `P0` sont toujours traitées en dernier.
- la commande des actions est exprimée par des identificateurs *(voir page 419)*.
- un maximum de 20 actions peuvent être affectées à chaque étape.
- la variable affectée à une action peut également être utilisée dans des actions d'autres étapes.
- vous pouvez utiliser la variable d'action en lecture et en écriture dans autant de sections du projet que vous désirez (affectation multiple !).
- les actions auxquelles un identificateur temporisé a été affecté ne peuvent être actives qu'une seule fois.
- seules les variables/adresses booléennes ou les éléments booléens de variables multi-éléments peuvent être utilisés comme variable d'action.
- les actions ont un nom unique.
  le nom de l'action est soit le nom de la variable d'action, soit le nom de la section d'action.

Variable d'action

Sont autorisées comme variables d'action :

- Adresse du type de données `BOOL`
  Une action peut être affectée à une sortie matérielle via une adresse. Dans ce cas, l'action peut être utilisée en tant que signal de validation d'une transition, en tant que signal d'entrée dans une autre section ou en tant que signal de sortie pour le matériel.
- Variable simple ou élément d'une variable multi-élément du type de donnée `BOOL`
  Une variable permet d'utiliser une action comme signal d'entrée dans une autre section.
  - Variable non localisée
    Pour les variables non localisées, l'action peut être utilisée en tant que signal de validation d'une transition et en tant que signal d'entrée dans une autre section.
  - Variable localisée
    Pour les variables localisées, l'action peut être utilisée comme signal de validation d'une transition, comme signal d'entrée dans une autre section et comme signal de sortie pour le matériel.

Noms d'action

Si vous utilisez une adresse ou une variable en tant qu'action, sa désignation sert de nom d'action (p.ex. %Q10.4, Variable1).

Si vous utilisez une section d'action en tant qu'action, le nom de section sert de nom d'action.

Les noms d'action (32 caractères maximum) doivent être uniques dans l'ensemble du projet, c.-à-d. qu'il ne doit pas exister d'autres transitions, variables, sections etc., ayant le même nom. Aucune distinction n'est faite entre l'écriture en majuscules et en minuscules. Le nom d'action doit répondre aux conventions sur les noms.

Section d'action

Présentation

Pour chaque action vous pouvez créer une section d'action. Il s'agit d'une section qui contient la logique de l'action et qui est automatiquement reliée à l'action.

Nom de la section d'action

Le nom de la section d'action est toujours identique au nom de l'action correspondante, voir *Noms d'action, page 417*.

Langages de programmation

Les langages de programmation possibles pour les sections d'action sont FBD, LD, IL et ST.

Caractéristiques des sections d'action

Les sections d'action ont les caractéristiques suivantes :

- elles peuvent comporter un nombre quelconque de sorties.
- les appels de sous-programmes ne sont possibles dans les sections d'action que si le mode Multitoken a été activé.
  Remarque : Les sous-programmes appelés ne sont **pas** soumis à la commande de la séquence, cela signifie que
  - l'identificateur affecté à la section d'action appelante n'a aucun effet sur le sous-programme,
  - le sous-programme reste actif, même à l'issue de la désactivation de l'étape appelante.
- les fonctions et blocs fonction ou procédures de diagnostic ne peuvent pas être utilisés dans les sections d'action.
- les sections d'action peuvent comporter de multiples réseaux.
- les sections d'action appartiennent à la section SFC dans laquelle elles ont été définies et peuvent dans cette section SFC (et toutes ses macro-sections) être affectées à un nombre quelconque d'actions.
- les sections d'action auxquelles un identificateur à durée a été affecté ne peuvent être actives qu'une seule fois.
- les sections d'action font partie de la section SFC dans laquelle elles ont été définies. Si la section SFC correspondante est supprimée, toutes les sections d'action de cette section SFC sont supprimées automatiquement.
- les sections d'action peuvent être appelées uniquement par des actions.

Qualificatif

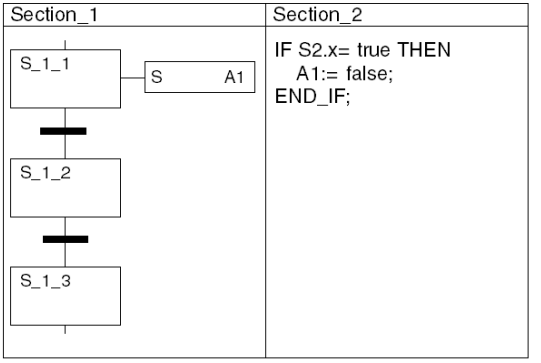
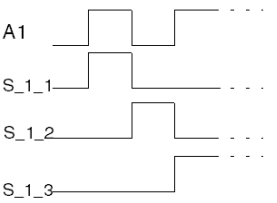
Présentation

A chaque connexion d'une action à une étape, il faut définir pour l'action un identificateur définissant la commande de l'action.

Identificateurs disponibles

Les identificateurs suivants sont disponibles :

Identificateur	Signification	Description
N / Aucun	Non mémorisé	Lorsque l'étape devient active, l'action passe à 1. Lorsque l'étape devient inactive, l'action tombe à 0.
R	Réinitialisation prioritaire	L'action définie dans une autre étape avec l'identificateur S est réinitialisée. En outre, il est possible d'éviter l'activation de n'importe quelle action. **Remarque :** les identificateurs sont déclarés automatiquement sans mémoire tampon. Cela signifie que leur valeur est remise à zéro après l'arrêt et le redémarrage du programme, p. ex. après une mise hors, puis sous tension. Si vous avez besoin d'une sortie avec une mémoire tampon, utilisez les blocs fonction RS ou SR de la bibliothèque de blocs standard.

Identificateur	Signification	Description
S	Définition (mémorisée)	Une action définie reste active lorsque l'étape correspondante devient inactive. L'action ne devient inactive que lorsqu'elle est réinitialisée dans une autre étape de la séquence en cours avec l'identificateur R. Cela signifie, par exemple, que la réinitialisation de l'action à partir d'une autre section n'est pas possible.   **Remarque :** les identificateurs sont déclarés automatiquement sans mémoire tampon. Cela signifie que leur valeur est remise à zéro après l'arrêt et le redémarrage du programme, p. ex. après une mise hors, puis sous tension. Si vous avez besoin d'une sortie avec une mémoire tampon, utilisez les blocs fonction RS ou SR de la bibliothèque de blocs standard. **Remarque :** Il est possible d'attribuer jusqu'à 100 actions dont l'identificateur est S par section SFC.
L	limité dans le temps	Lorsque l'étape devient active, l'action le devient également. Lorsque la durée que vous avez définie pour l'action est écoulée, l'action repasse à 0, même si l'étape est encore active. L'action passe également à 0 lorsque l'étape devient inactive. **Remarque :** pour cet identificateur, il faut de plus définir une durée du type TIME.

Identificateur	Signification	Description
D	retardé	Lorsque l'étape devient active, le temporisateur interne est lancé et l'action est définie sur 1 après la durée définie manuellement pour l'action. Ensuite, lorsque l'étape devient inactive, l'action devient également inactive. Si l'étape devient inactive avant écoulement du temps interne, l'action ne devient pas active. **Remarque :** pour cet identificateur, il faut de plus définir une durée du type TIME.
P	Impulsion	Lorsque l'étape devient active, l'action passe à 1 et le reste pendant un cycle de programme, indépendamment du fait que l'étape reste active ou non.
DS	retardé et mémorisé	Lorsque l'étape devient active, le temporisateur interne est lancé et lorsque la durée définie est écoulée, l'action devient active. L'action ne redevient inactive que lorsqu'elle est réinitialisée dans une autre étape avec l'identificateur R. Si l'étape devient inactive avant écoulement du temps interne, l'action ne devient pas active. **Remarque :** pour cet identificateur, il faut de plus définir une durée du type TIME.
P1	Impulsion (front montant)	Lorsque l'étape devient active (front 0->1), l'action passe à 1 et le reste pendant un cycle de programme, indépendamment du fait que l'étape reste active ou non. **Remarque :** Indépendamment de leur position dans la zone de liste des actions, les actions avec l'identificateur P1 sont toujours exécutées en premier, voir également *Action, page 416*.
P0	Impulsion (front descendant)	Lorsque l'étape devient inactive (front 1->0), l'action passe à 1 et le reste pendant un cycle de programme. **Remarque :** Indépendamment de leur position dans la zone de liste des actions, les actions avec l'identificateur P0 sont toujours exécutées en dernier, voir également *Action, page 416*.

13.4 Transition et section de transition

Objet de ce sous-chapitre

Ce sous-chapitre décrit les objets transition et les sections de transition du diagramme fonctionnel en séquence SFC.

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Transition

Présentation

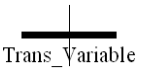
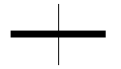
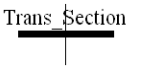
Une transition indique la condition à laquelle le contrôle d'une ou de plusieurs étapes précédant la transition passe à une ou plusieurs étapes subséquentes, le long de la liaison correspondante.

Condition de transition

A chaque transition est affectée une condition de transition du type de donnée `BOOL`.

Sont autorisés comme conditions de transition :
- une adresse (entrée ou sortie),
- une variable (entrée ou sortie),
- un libellé ou
- une section de transition *(voir page 425)*.

Le type de condition de transition détermine la position de son nom.

Condition de transition	Position du nom
• Adresse • Variable	
• Libellé	
• Section de transition	

Noms de transition

Si vous utilisez une adresse ou une variable comme condition de transition, sa désignation est utilisée comme nom de transition (p. ex. `%I10.4`, `Variable1`).

Si vous utilisez une section de transition comme condition de transition, le nom de section est utilisé comme nom de transition.

Les noms de transition (32 caractères maximum) doivent être uniques dans l'ensemble du projet, c.-à-d. qu'il ne doit pas exister d'autres transitions, variables, sections etc. (à l'exception de la section de transition correspondante) ayant le même nom. Aucune distinction n'est faite entre l'écriture en majuscules et en minuscules. Le nom de transition doit répondre aux conventions sur les noms.

Validation d'une transition

Une transition est validée lorsque les étapes qui la précèdent directement sont actives. Les transitions, dont les étapes qui les précèdent directement ne sont pas actives, ne sont normalement pas évaluées.

NOTE : Si aucune condition de transition n'est définie, la transition ne sera jamais active.

Déclenchement d'une transition

Le déclenchement d'une transition s'effectue lorsque la transition est validée et que la condition de transition correspondante est vraie.

Le déclenchement d'une transition entraîne la désactivation (réinitialisation) de toutes les étapes qui la précèdent directement et qui sont connectées à la transition. Ensuite, toutes les étapes qui la suivent directement sont activées.

Temps de déclenchement d'une transition

Théoriquement, le temps de déclenchement (temps de commutation) d'une transition peut être considéré comme étant le plus court possible, mais il ne peut jamais prendre la valeur zéro. Le temps de déclenchement d'une transition dure au moins le temps d'un cycle de programme.

Section de transition

Présentation

Une section de transition peut être créée pour chaque transition. Il s'agit d'une section qui contient la logique de la condition de transition et qui est automatiquement reliée à la transition.

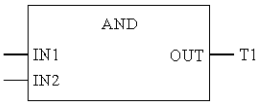
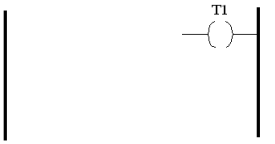
Nom de la section de transition

Le nom de la section de transition est toujours identique au nom de la transition correspondante, voir *Noms de transition, page 423*.

Langages de programmation

Les langages de programmation des sections de transition sont FBD , LD , IL et ST.

Réseaux conseillés pour les sections de transition :

Langage	Réseau conseillé	Description
FBD		Le réseau conseillé contient un bloc `ET` comportant 2 entrées et dont la sortie booléenne est liée à une variable qui porte le nom de la section de transition. Le bloc proposé peut être relié ou, si vous ne le désirez pas, être supprimé.
LD		Le réseau proposé contient une bobine reliée à une variable portant le nom de la section de transition. La bobine proposée peut être reliée ou, si vous ne la désirez pas, être supprimée.

IL	-	Le réseau proposé est vide.
		Le seul contenu admis consiste en la création d'une logique booléenne. L'affectation du résultat logique à la sortie (variable de transition) est effectuée automatiquement, c-à-d. l'instruction mémoire `ST` n'est pas admise.
		Exemple :
		`LD A`
		`AND B`

Langage	Réseau conseillé	Description

ST	-	Le réseau proposé est vide.
		Le seul contenu admis consiste à créer une logique booléenne sous forme d'expression (imbriquée). L'affectation du résultat logique à la sortie (variable de transition) est effectuée automatiquement, c-à-d. l'instruction d'affectation = n'est pas admise. L'expression ne se termine pas par un point-virgule (`;`).
		Exemple :
		`A AND B`
		ou
		`A AND (WORD_TO_BOOL (B))`

Caractéristiques des sections de transition

Les sections de transition ont les caractéristiques suivantes :

- les sections de transition n'ont qu'une seule sortie (la variable de transition) et son type de donnée est `BOOL`. Le nom de cette variable est identique au nom de la section de transition.
- la variable de transition ne doit être utilisée en écriture qu'une seule fois.
- la variable de transition peut être lue à n'importe quelle position du projet.
- seules des fonctions peuvent être utilisées et non pas des blocs fonction ni des procédures.
- dans LD, seule une bobine peut être utilisée.
- il n'existe qu' un seul réseau, c.-à-d. que toutes les fonctions utilisées sont reliées entre elles directement ou indirectement.
- les sections de transition ne peuvent être utilisées qu'une seule fois.
- les sections de transition font partie de la section SFC dans laquelle elles ont été définies. Si la section SFC correspondante est supprimée, toutes les sections de transition de ces sections SFC sont supprimées automatiquement.
- les sections de transition ne peuvent être appelées que par une transition.

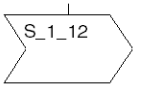
13.5 Saut

Saut

Généralités

Les sauts permettent de représenter des liaisons dirigées qui ne sont pas tracées entièrement.

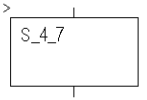
Représentation d'un saut :



Caractéristiques d'un saut

Un saut a les caractéristiques suivantes :

- plusieurs sauts peuvent avoir une même étape pour cible.
- selon la norme CEI 61131-3, les sauts dans une séquence en ET *(voir page 431)* ou depuis une séquence en ET ne sont pas autorisés.
  si vous voulez les utiliser tout de même, vous devez les valider explicitement.
- dans le cadre des sauts, on distingue les sauts de séquence *(voir page 436)* et les boucles de séquence *(voir page 437)*.
- la cible du saut est munie du symbole du saut (>).



Nom de saut

Les sauts ne possèdent pas de nom en propre. Au lieu de cela, le nom de l'étape cible (cible du saut) est indiqué sur le symbole du saut.

13.6 Liaison

Liaison

Présentation

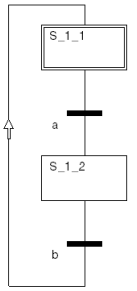
Les liaisons relient les étapes et les transitions, les transitions et les étapes etc.

Caractéristiques des liaisons

Les liaisons ont les caractéristiques suivantes :

- il est impossible d'établir une liaison entre des objets de même type (étape avec étape, transition avec transition, etc.).
- une liaison peut être établie entre :
  - sorties d'objets non reliées et
  - entrées d'étape non reliées et reliées
    (cela signifie que une liaison multiple d'entrées d'étapes est possible)
- le chevauchement entre des liaisons et d'autres objets SFC (étape, transition, saut, etc.) n'est pas possible.
- les chevauchements entre liaisons sont possibles.
- Le croisement inter-liaison est possible et est indiqué sous forme de liaison "interrompue" :

- les liaisons se composent de segments verticaux et horizontaux.
- l'évolution de l'état logique dans une séquence est généralement de haut en bas. Cependant pour permettre la mise en oeuvre de boucles, les liaisons vers une étape peuvent également se faire de bas en haut. Ceci est valable pour les liaisons de transitions, de divergences en ET et de convergences en OU vers une étape. Dans ce cas, le sens de la liaison est représentée par une flèche.



- pour les liaisons, on distingue les sauts de séquence *(voir page 436)* et les boucles de séquence *(voir page 437)*.

13.7 Divergences et convergences

Objet de ce sous-chapitre

Ce sous-chapitre décrit les objets de convergence et de divergence du diagramme fonctionnel en séquence SFC.

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Divergence en OU et convergence en OU

Présentation

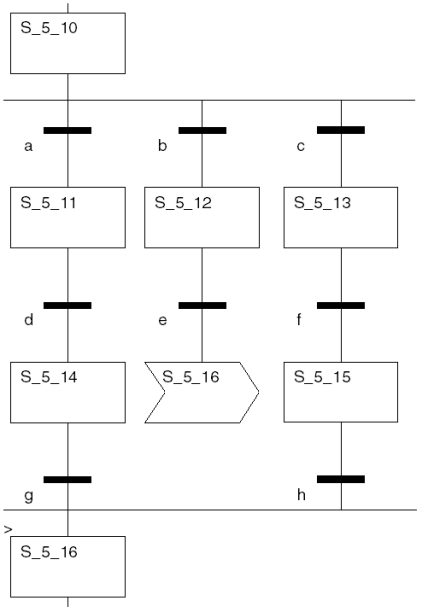
La divergence en OU permet de programmer des divergences à conditions au sein du flux de contrôle de la structure SFC.

Dans les divergences en OU, une étape est suivie d'autant de transitions sous la ligne horizontale qu'il y a d'enchaînements différents.

A l'aide de convergences en OU ou de sauts *(voir page 427)*, toutes les branches alternatives convergent vers une branche unique dans laquelle le traitement se poursuit.

Exemple d'une séquence en OU

Exemple d'une séquence en OU



Caractéristiques d'une séquence en OU

Les caractéristiques d'une séquence en OU dépend en grande partie du fait que la commande de séquence soit mise en oeuvre en jeton unique ou jetons multiples.

Voir

- Caractéristiques d'une séquence en OU en jeton unique *(voir page 435)*
- Caractéristiques d'une séquence en OU en jetons multiples *(voir page 447)*

Divergence en ET et convergence en ET

Présentation

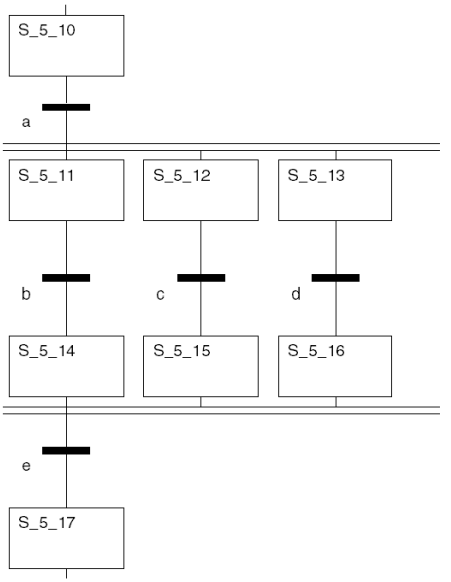
Avec les divergences en ET, la commutation d'une seule transition active plusieurs étapes (branches, 32 au maximum) en parallèle. Elles sont exécutées de gauche à droite. Après cette activation commune, les différentes branches sont exécutées indépendamment.

Conformément à la norme CEI 61131-1, toutes les branches d'une divergence en ET sont regroupées par une convergence en ET. La transition après une convergence en ET est évaluée lorsque toutes les étapes immédiatement précédentes de la convergence en ET ont été validées.

Si une divergence en ET doit être regroupée par une convergence en OU, ceci n'est possible qu'en mode jetons multiples *(voir page 450)*.

Exemple d'une séquence en ET

Exemple d'une séquence en ET



Caractéristiques d'une séquence en ET

voir

- Caractéristiques d'une séquence en ET dans un jeton unique *(voir page 435)*
- Caractéristiques d'une séquence en ET dans un jeton multiple *(voir page 447)*

13.8 Objet texte

Objet texte

Présentation

En langage SFC, il est possible de positionner du texte sous forme d'objets texte. La taille de cet objet texte dépend de la longueur du texte. Cet objet texte a au moins la taille d'une cellule et peut, selon la taille du texte, être étendu sur des cellules complémentaires dans le sens vertical ou horizontal. Les objets de texte peuvent se recouvrir avec d'autres objets SFC.

13.9 Jeton unique

Objet de ce sous-chapitre

Ce sous-chapitre décrit le mode "jeton unique" pour les commandes de séquence.

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Jeton unique d'ordre d'exécution

Description

Les règles suivantes s'appliquent aux jetons uniques :

- La situation d'origine est définie par l'étape initiale. La commande de séquence ne contient qu'une seule étape initiale.
- une seule étape est active à la fois dans une séquence. La seule exception est pour les divergences en ET dans lesquelles une étape est active par divergence.
- Les évolutions des états de signaux actifs se produisent le long des liaisons dirigées et sont déclenchés par la commutation d'une ou de plusieurs transitions. La séquence d'une chaîne va dans le sens des liaisons dirigées et se déroule de la partie inférieure de l'étape précédente à la partie supérieure de l'étape suivante.
- Une transition est validée lorsque les étapes qui la précèdent directement sont actives. Les transitions, dont les étapes qui les précèdent directement ne sont pas actives, ne sont normalement pas évaluées.
- Le déclenchement d'une transition s'effectue lorsque la transition est validée et que la condition de transition correspondante est vraie.
- Le déclenchement d'une transition entraîne la désactivation (réinitialisation) de toutes les étapes qui la précèdent directement et qui sont connectées à la transition. Ensuite, toutes les étapes qui la suivent directement sont activées.
- Lorsque plusieurs conditions de transition d'une série d'étapes consécutives sont remplies, la scrutation est effectué à une cadence d'une étape par cycle.
- Les étapes ne peuvent pas être activées ni désactivées par d'autres sections non SFC.
- Vous pouvez utiliser des macroétapes.
- Une seule branche est active à la fois dans une divergence en OU. Le résultat des conditions de transition détermine quelle branche est exécutée en fonction des transitions qui suivent la divergence en OU. Si une condition de transition est vraie, les transitions restantes ne sont plus exécutées. La branche qui contient la transition remplie devient active. Les divergences répondent à une priorité de gauche à droite. Toutes les divergences en OU sont regroupées à leur issue par une convergence en OU ou par des sauts.
- pour les divergences en ET, la commutation d'une seule transition active plusieurs étapes (branches) en parallèle. Après cette activation commune, les différentes branches sont exécutées indépendamment. Toutes les branches en ET sont regroupées à leur issue par une convergence en ET. Il n'est pas possible d'effectuer des sauts vers ou à partir d'une branche en ET.

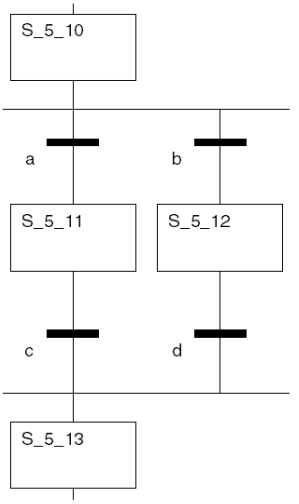
Séquence en OU

Séquence en OU

Selon la norme CEI 61131-3, seule une transition peut être vraie à la fois (1 parmi n choix). Le résultat des conditions de transition déterminera quelle branche sera exécutée en fonction des transitions qui suivent la divergence en OU. Les transitions de la divergence sont exécutées de gauche à droite. Si une condition de transition est vraie, les transitions restantes ne seront plus exécutées. La branche qui contient la transition remplie devient active. Il en résulte une priorité de gauche à droite pour les divergences.

Si aucune des transitions n'est vraie, l'étape validée à ce moment-là reste valide.

Séquence en OU :



Si...	Alors
Si S_5_10 est active et que la condition de transition a est vraie (indépendamment de b),	S_5_10 passe à S_5_11.
Si S_5_10 est active, que la condition de transition b est vraie et que a) est fausse,	S_5_10 passe à S_5_12.

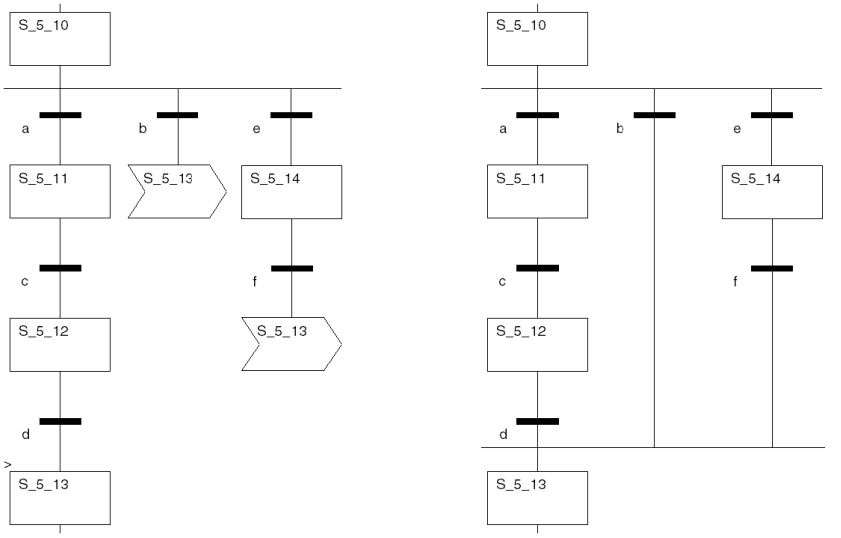
Sauts de séquence et boucles de séquence

Saut de séquence

Un saut de séquence est un cas particulier de la divergence en OU dans lequel quelques étapes de la séquence sont ignorées.

Un saut de séquence peut être réalisé avec des sauts ou avec des liaisons.

Saut de séquence :



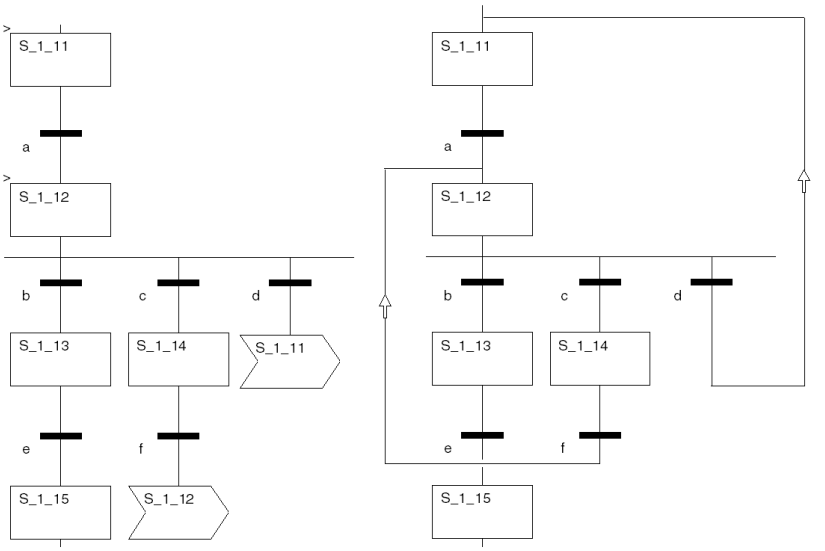
Si...	Alors
Si la condition de transition a est vraie,	la séquence passe de S_5_10 à S_5_11 puis S_5_12 et S_5_13.
Si la condition de transition b est vraie,	la séquence effectue un saut direct de S_5_10 vers S_5_13.
Si la condition de transition e est vraie,	la séquence passe de S_5_10 à S_5_14 et S_5_13.

Boucle de séquence

Une séquence en boucle est un cas spécial de la dérivation en OU pour laquelle une ou plusieurs branches reconduisent à une étape précédente.

Une séquence en boucle peut être réalisée avec des sauts ou avec des liaisons.

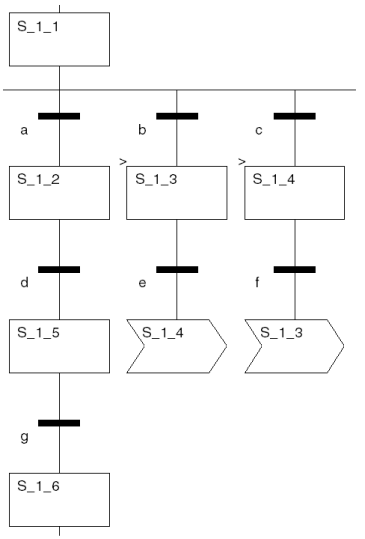
Boucle de séquence :

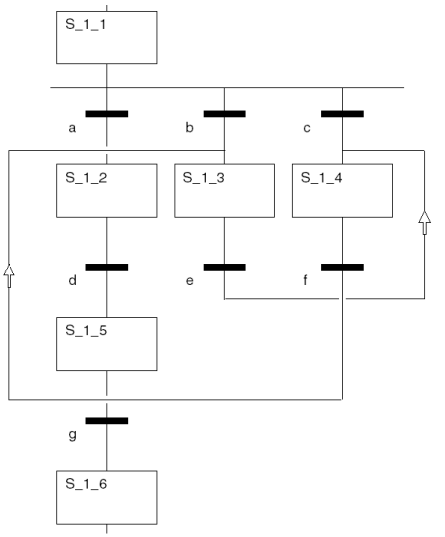


Si...	**Alors**
Si la condition de transition `a` est vraie,	la séquence passe de `S_1_11` à `S_1_12`.
Si la condition de transition `b` est vraie,	la séquence passe de `S_1_12` à `S_1_13`.
Si la condition de transition `b` est fausse et que `c` est vraie,	`S_1_12` est suivie de `S_1_14`.
Si la condition de transition `f` est vraie,	un saut est réalisé à partir de `S_1_14` jusque `S_1_12`.
La boucle de l'étape `S_1_12` via les conditions de transition `c` et `f` et le bouclage vers `S_1_12` est répétée jusqu'à ce que la condition de transition `b` soit vraie ou que `c` soit fausse et `d` vraie.	
Si les conditions de transition `b` et `c` sont fausses et que `d` est vraie,	`S_1_12` effectue un saut en arrière direct vers `S_1_11`.
La boucle de `S_1_11` vers `S_1_12` et retour vers `S_1_11` via les conditions de transition `a` et `d` est répétée jusqu'à ce que la condition de transition `b` ou `c` devienne vraie.	

Les séquences en boucle sans fin ne sont pas admises dans une séquence alternative.

Séquence en boucle sans fin :





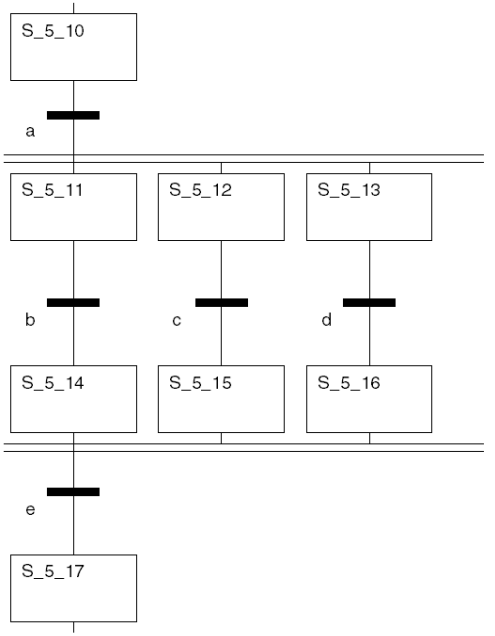
Si...	**Alors**
Si la condition de transition `b` est vraie,	`S_1_1` passe à `S_1_3`.
Si la condition de transition `e` est vraie,	un saut est exécuté vers `S_1_4`.
Si la condition de transition `f` est vraie,	un saut est exécuté vers `S_1_3`.
La boucle de `S_1_3` via la condition de transition `e` vers S_1_4 via la condition de transition `f` et le saut de retour vers `S_1_3` est alors répétée sans fin.	

Séquences en ET

Séquences en ET

Pour les divergences en ET, la validation d'une seule transition active plusieurs étapes (branches, 32 au maximum) en parallèle. Ceci est valable aussi bien pour le jeton unique que pour les jetons multiples.

Exécution des séquences en ET :

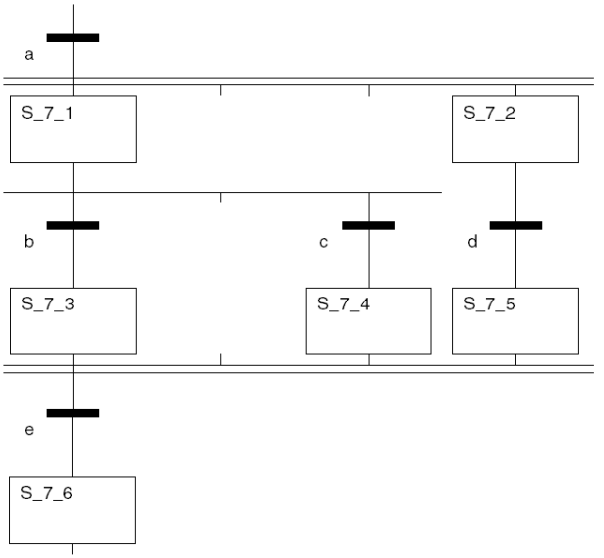


Si...	Alors
Si S_5_10 est active et que la condition de transition a, faisant partie de la transition commune, est également vraie,	S_5_10 passe à S_5_11, S_5_12 et S_5_13.
Si les étapes S_5_11, S_5_12et S_5_13 sont activées,	les séquences sont exécutées indépendamment.
Si S_5_14, S_5_15 et S_5_16 sont actives simultanément et que la condition de transition e faisant partie de la transition commune est vraie,	S_5_14, S_5_15 et S_5_16 passent à S_5_17.

Utilisation d'une divergence en OU dans une séquence en ET.

Si vous utilisez une seule divergence en OU dans une séquence en ET, ceci provoque le blocage de la séquence dans le cas des jetons uniques.

Utilisation d'une divergence en OU dans une séquence en ET :



Si...	**Alors**
Si la condition de transition `a` est vraie,	la séquence passe à `S_7_1` et `S_7_2`.
Si les étapes `S_7_1` et `S_7_2` sont activées,	les séquences sont exécutées indépendamment.
Si la condition de transition `d` est vraie,	`S_7_5` est exécutée.
Si la condition de transition `b` est vraie et que `c` est fausse,	la séquence passe à `S_7_3` est exécutée.
`S_7_3`, `S_7_4` et `S_7_5` étant reliées par une convergence en ET, `S_7_6` ne peut pas être exécutée puisque `S_7_3` et `S_7_4` ne peuvent jamais devenir actives en même temps. (Soit `S_7_3` a été activée par la condition de transition `b`, soit `S_7_4` a été activée par la condition de transition `c`, mais jamais les deux en même temps.) C'est pourquoi `S_7_3`, `S_7_4` et `S_7_5` ne peuvent pas non plus devenir actives en même temps. La séquence se bloque. Le même problème survient si lors de l'entrée dans la divergence en OU la condition de transition `b` est fausse et `c` est vraie.	

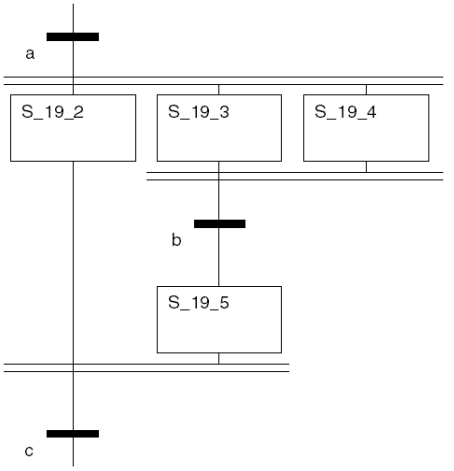
Sélection d'une séquence en ET asymétrique

Présentation

Selon la norme CEI 61131-3, une divergence en ET doit toujours être fermée par une convergence en ET. Le nombre de divergences en ET ne doit cependant pas nécessairement correspondre au nombre des convergences en ET.

Nombre supérieur de convergences

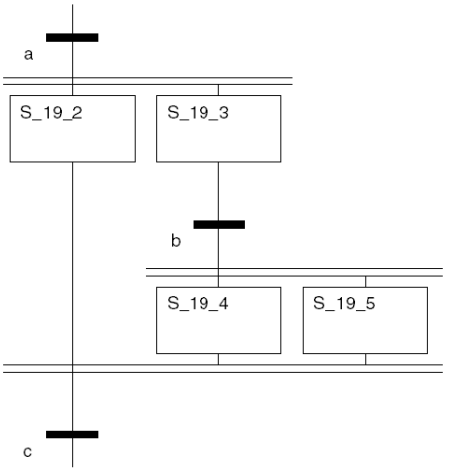
Séquence avec 1 divergence en ET et 2 convergences en ET



Si...	Alors
Si la condition de transition `a` est vraie,	`S_19_2`, `S_19_3` et `S_19_4` sont exécutées.
Si les étapes `S_19_2`, `S_19_3` et `S_19_4` sont activées,	les séquences sont exécutées indépendamment.
Si la condition de transition `b` est vraie,	`S_19_5` est exécutée.
Si les étapes `S_19_2` et `S_19_5` sont actives et que la condition de transition `c` est vraie,	la séquence en ET est abandonnée.

Nombre supérieur de divergences

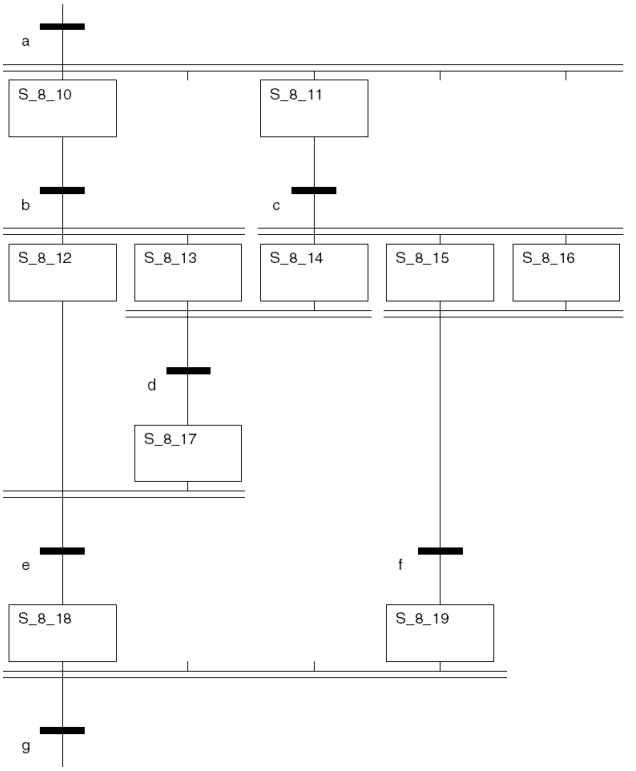
Séquence avec 2 divergences en ET et 1 convergence en ET



Si...	**Alors**
Si la condition de transition `a` est vraie,	`S_19_2` et `S_19_3` sont exécutées.
Si les étapes `S_19_2` et `S_19_3` sont activées,	les séquences sont exécutées indépendamment.
Si la condition de transition `b` est vraie,	`S_19_4` et `S_19_5` sont exécutées.
Si les étapes `S_19_4` et `S_19_5` sont activées,	les séquences sont exécutées indépendamment.
Si les étapes `S_19_2`, `S_19_4` et `S_19_5` sont actives et que la condition de transition `c` est vraie,	la séquence en ET est abandonnée.

Séquences en ET imbriquées

Séquences en ET imbriquées :



Si...	**Alors**
Si la condition de transition a est vraie,	S_8_10 et S_8_11 sont exécutées.
Si la condition de transition b est vraie,	S_8_12 et S_8_13 sont exécutées.
Si la condition de transition c est vraie,	S_8_14, S_8_15 et S_8_16 sont exécutées.
Si les étapes S_8_13 et S_8_14 sont actives et que la condition de transition d est vraie,	S_8_17 est exécutée.
Si les étapes S_8_12 et S_8_17 sont actives et que la condition de transition e est vraie,	S_8_18 est exécutée.
...	...

13.10 Jetons multiples

Objet de ce sous-chapitre

Ce sous-chapitre décrit le mode "jetons multiples" pour les commandes de séquence.

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Ordre d'exécution à plusieurs jetons

Description

Les règles suivantes s'appliquent aux jetons multiples :

- la situation de départ est définie par un nombre défini d'étapes initiales (de 0 à 100).
- dans la séquence, vous pouvez définir librement un nombre d'étapes qui seront simultanément actives.
- les évolutions des états de signaux actifs se produisent le long des liaisons dirigées et sont déclenchés par la commutation d'une ou de plusieurs transitions. La séquence d'une chaîne va dans le sens des liaisons dirigées et se déroule de la partie inférieure de l'étape précédente à la partie supérieure de l'étape suivante.
- une transition est validée lorsque les étapes qui la précèdent directement sont actives. Les transitions dont les étapes qui les précèdent directement ne sont pas actives, ne sont pas évaluées.
- le déclenchement d'une transition s'effectue lorsque la transition est validée et que la condition de transition correspondante est vraie.
- le déclenchement d'une transition entraîne la désactivation (réinitialisation) de toutes les étapes qui la précèdent directement et qui sont connectées à la transition. Ensuite, toutes les étapes qui la suivent directement sont activées.
- lorsque plusieurs conditions de transition d'une série d'étapes consécutives sont remplies, la scrutation est effectué à une cadence d'une étape par cycle.
- les étapes et macro-étapes peuvent être activées ou désactivées par d'autres sections non SFC ou par des opérations utilisateur.
- si une étape active est simultanément activée et désactivée, l'étape reste active.
- vous pouvez utiliser des macroétapes. Les sections de macro-étapes peuvent également comporter des étapes initiales.
- dans les divergences en OU, plusieurs branches peuvent être actives simultanément. Le résultat des conditions de transition des transitions qui suivent la divergence en OU déterminera quelle branche sera exécutée. Les transitions des branches sont exécutées en parallèle. Les branches dont la transition est remplie deviennent actives. Les branches en OU ne doivent pas être regroupées à leur issue par une convergence en OU ou par des sauts.
- si vous voulez mettre en oeuvre des sauts vers ou depuis une branche en ET, vous pouvez valider une option correspondante. Dans ce cas, les branches en ET ne doivent pas être regroupées à leur issue par une convergence en ET.
- des appels de sous-programmes peuvent être utilisés dans une section d'action.

- vous pouvez générer des jetons multiples à l'aide des éléments suivants :
  - étapes initiales multiples,
  - divergences en OU ou en ET non refermées,
  - sauts associés à des séquences en OU ou en ET,
  - activation d'étapes à l'aide du bloc de commande SFC `SETSTEP` depuis une section non-SFC ou à l'aide d'instructions de commande SFC.
- les jetons sont terminés par les actions suivantes :
  - rencontre simultanée de deux jetons ou plus dans une même étape,
  - désactivation d'étapes à l'aide du bloc de commande SFC `RESETSTEP` depuis une section non-SFC ou à l'aide d'instructions de commande SFC.

Séquence en OU

Séquence en OU

Dans le cas des jetons multiples, l'utilisateur peut définir le comportement d'évaluation des conditions de transitions dans les divergences en OU.

Les paramétrages possibles sont les suivants :

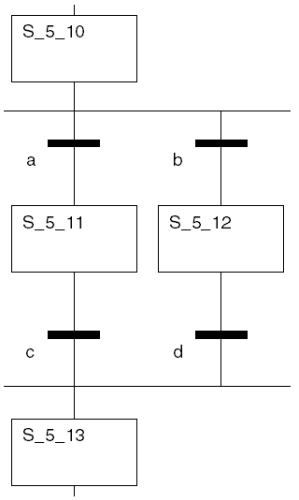
- Exécution de gauche à droite avec arrêt après la première transition active (choix de 1 parmi n). Ceci correspond au comportement des séquences en OU pour les Jetons uniques *(voir page 435)*.
- Exécution en parallèle de toutes les transitions de la divergence en OU (choix de x parmi n)

Choix de x parmi n

Pour le cas des jetons multiples, plusieurs transitions peuvent être validées parallèlement (choix de x parmi n). Le résultat des conditions de transition déterminera quelle branche sera exécutée en fonction des transitions qui suivent la divergence en OU. Les transitions des branches sont toutes exécutées. Toutes les branches dont la transition est remplie deviennent actives.

Si aucune des transitions n'est vraie, l'étape validée à ce moment-là reste valide.

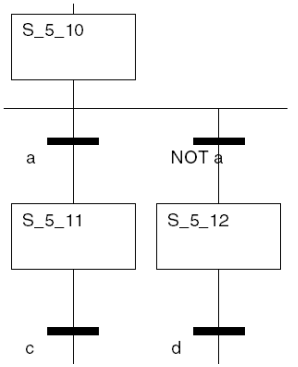
Choix de x parmi n :



<table>
<tr><th colspan="2">Si...</th><th colspan="2">Alors</th></tr>
<tr><td colspan="2">Si S_5_10 est active et que la condition de transition a est vraie et que b est fausse,</td><td colspan="2">S_5_10 passe à S_5_11.</td></tr>
<tr><td colspan="2">Si S_5_10 est active et que la condition de transition a est fausse et b est vraie,</td><td colspan="2">S_5_10 passe à S_5_12.</td></tr>
<tr><td colspan="2">Si S_5_10 est active et que les conditions de transition a et b sont vraies,</td><td colspan="2">S_5_10 passe à S_5_11 et à S_5_12.</td></tr>
<tr><td colspan="4">L'activation en parallèle des deux branches en OU génère un deuxième jeton. Les deux jetons fonctionnent maintenant en parallèle, c.-à-d. S_5_11 et S_5_12 sont actives simultanément.</td></tr>
<tr><th colspan="2">Jeton 1 (S_5_11)</th><th colspan="2">Jeton 2 (S_5_12)</th></tr>
<tr><th>Si...</th><th>Alors</th><th>Si...</th><th>Alors</th></tr>
<tr><td>Si la condition de transition c est vraie,</td><td>S_5_11 passe à S_5_13.</td><td>Si la condition de transition d est vraie,</td><td>S_5_12 passe à S_5_13.</td></tr>
<tr><td colspan="4">Si S_5_13 est encore active en raison de l'activation de la condition de transition c (jeton 1), le jeton 2 s'arrête et la séquence est exécutée à nouveau en jeton unique. Si S_5_13 n'est plus active (jeton 1), elle est réactivée par le jeton 2 et les deux jetons continuent à être exécutés en parallèle (jetons multiples).</td></tr>
</table>

Si vous désirez que les divergences en OU ne soient validées que de manière exclusive dans ce mode, vous devez le définir explicitement dans la logique de transition.

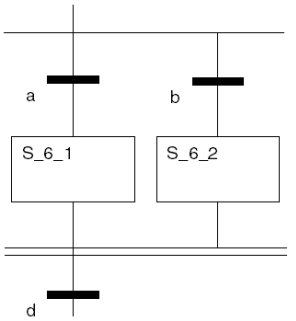
Exemple :



Terminaison d'une divergence en OU par une convergence en ET

Si une divergence en OU est refermée par une convergence en ET, ceci peut déclencher un blocage de la séquence.

Terminaison d'une divergence en OU par une convergence en ET :



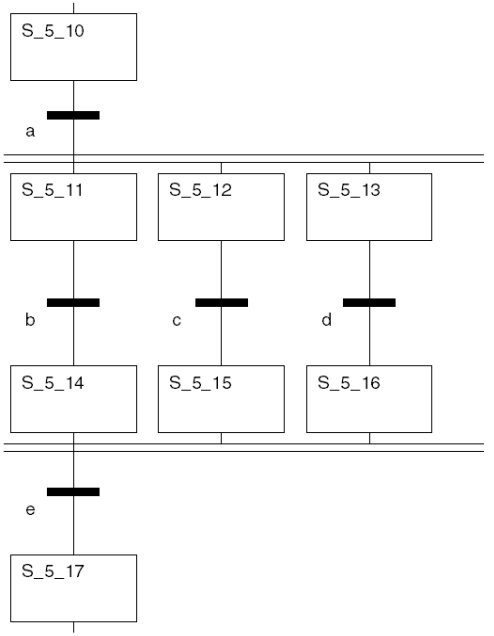
Si...	Alors
Si la condition de transition `a` est vraie et que `b` est fausse,	la séquence passe à `S_6_1`.
`S_6_1` et `S_6_2` étant reliées par une convergence en ET, il n'est pas possible de quitter la divergence parce que `S_6_1` et `S_6_2` ne peuvent jamais devenir actives en même temps. (Soit `S_6_1` a été activée par la condition de transition `a` soit `S_6_2` a été activée par la condition de transition `b`.) C'est pourquoi `S_6_1` et `S_6_2` ne peuvent pas non plus devenir actives en même temps. La séquence se bloque. Ce blocage peut être levé p.ex. par un deuxième jeton décalé dans le temps, exécuté via la transition `b`.	

Séquences en ET

Séquences en ET

Pour les divergences en ET, la validation d'une seule transition active plusieurs étapes (branches, 32 au maximum) en parallèle. Ceci est valable aussi bien pour le jeton unique que pour les jetons multiples.

Exécution des séquences en ET :

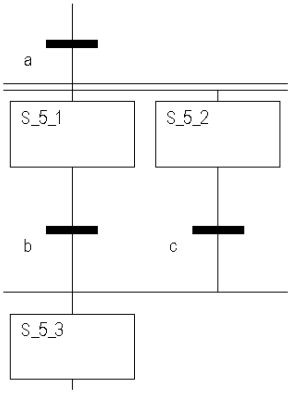


Si...	Alors
Si S_5_10 est active et que la condition de transition a, faisant partie de la transition commune, est également vraie,	S_5_10 passe à S_5_11, S_5_12 et S_5_13.
Si les étapes S_5_11, S_5_12et S_5_13 sont activées,	les séquences sont exécutées indépendamment.
Si S_5_14, S_5_15 et S_5_16 sont actives simultanément et que la condition de transition e faisant partie de la transition commune est vraie,	S_5_14, S_5_15 et S_5_16 passent à S_5_17.

Terminaison d'une divergence en ET par une convergence en OU

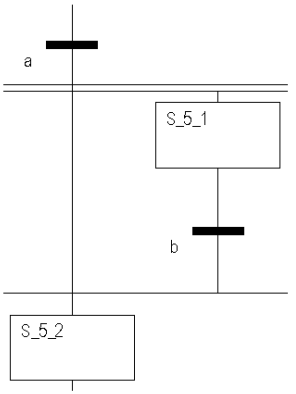
Pour les jetons multiples, il est possible de terminer les divergences en ET par une convergence en OU au lieu d'une convergence en ET.

Terminaison d'une séquence en ET par une divergence en OU (variante 1)



Si...		**Alors**	
Si la condition de transition `a` est vraie,		`S_5_1` et `S_5_2` sont exécutées.	
Si les étapes `S_5_1` et `S_5_2` sont activées,		les séquences sont exécutées indépendamment.	
Si la condition de transition `b` est vraie et que `c` est fausse,		`S_5_3` est exécutée.	
L'exécution depuis la séquence en ET via la divergence en OU génère un deuxième jeton. Les deux jetons fonctionnent maintenant en parallèle, c.-à-d. `S_5_2` et `S_5_3` sont actives simultanément.			
Jeton 1 (S_5_3)		**Jeton 2 (S_5_2)**	
Si...	**Alors**	**Si...**	**Alors**
L'étape `S_5_3` est active.		L'étape `S_5_2` est active.	
		Si la condition de transition `c` est vraie,	`S_5_3` est exécutée.
Si `S_5_3` est encore active (jeton 1), le jeton 2 s'arrête et la séquence est exécutée à nouveau en jeton unique. Si `S_5_3` n'est plus active (jeton 1), elle est réactivée par le jeton 2 et les deux jetons continuent à être exécutés en parallèle (jetons multiples).			

Terminaison d'une séquence en ET par une divergence en OU (variante 2)

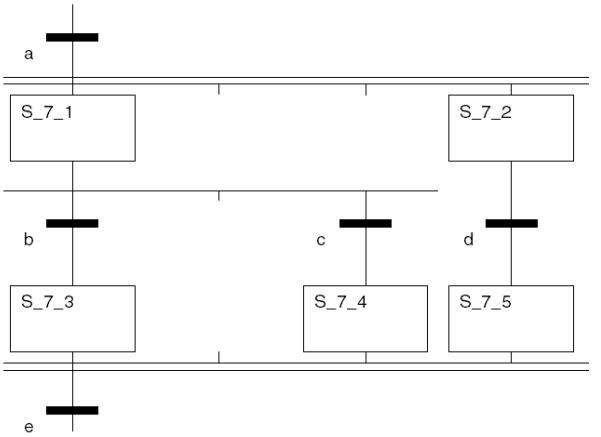


<table>
<tr><th colspan="2">Si...</th><th colspan="2">Alors</th></tr>
<tr><td colspan="2">Si la condition de transition a est vraie,</td><td colspan="2">S_5_1 et S_5_2 sont exécutées.</td></tr>
<tr><td colspan="4">L'exécution depuis la séquence en ET via la divergence en OU génère un deuxième jeton. Les deux jetons fonctionnent maintenant en parallèle, c.-à-d. S_5_1 et S_5_2 sont actives simultanément.</td></tr>
<tr><th colspan="2">Jeton 1 (S_5_2)</th><th colspan="2">Jeton 2 (S_5_1)</th></tr>
<tr><th>Si...</th><th>Alors</th><th>Si...</th><th>Alors</th></tr>
<tr><td colspan="2">L'étape S_5_2 est active.</td><td colspan="2">L'étape S_5_1 est active.</td></tr>
<tr><td colspan="2"></td><td>Si la condition de transition b est vraie,</td><td>S_5_2 est exécutée.</td></tr>
<tr><td colspan="4">Si S_5_2 est encore active (jeton 1), le jeton 2 s'arrête et la séquence est exécutée à nouveau en jeton unique.
Si S_5_2 n'est plus active (jeton 1), elle est réactivée par le jeton 2 et les deux jetons continuent à être exécutés en parallèle (jetons multiples).</td></tr>
</table>

Utilisation d'une divergence en OU dans une séquence en ET.

Si une divergence en OU est terminée par une séquence en ET, ceci peut déclencher un blocage de la séquence.

Utilisation d'une divergence en OU dans une séquence en ET :



Si...	**Alors**
Si la condition de transition `a` est vraie,	la séquence passe à `S_7_1` et `S_7_2`.
Si les étapes `S_7_1` et `S_7_2` sont activées,	les séquences sont exécutées indépendamment.
Si la condition de transition `d` est vraie,	`S_7_5` est exécutée.
Si la condition de transition `b` est vraie,	la séquence passe à `S_7_3`.
`S_7_3`, `S_7_4` et `S_7_5` étant reliées par une convergence en ET, il n'est pas possible de quitter la séquence en ET car `S_7_3` et `S_7_4` ne peuvent pas devenir actives en même temps. (Soit `S_7_3` a été activée par la condition de transition `b` ,soit `S_7_4` a été activée par la condition de transition `c`.) C'est pourquoi `S_7_3`, `S_7_4` et `S_7_5` ne peuvent pas non plus devenir actives en même temps. La séquence se bloque. Ce blocage peut être levé p.ex. par un deuxième jeton décalé dans le temps, exécuté via la transition `c`.	

Saut dans une séquence en ET

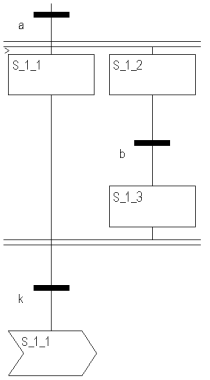
Description

Pour les jetons multiples, vous pouvez valider au choix le saut dans une séquence en ET ou hors d'une séquence en ET.

Un saut dans une séquence en ET n'active pas toutes les branches. Suite à la convergence en ET la transition n'étant évaluée que lorsque toutes les étapes qui la précècent directement ont été validées, vous ne pouvez plus quitter la séquence en ET, elle reste bloquée.

Saut dans une séquence en ET

Saut dans une séquence en ET



Si...	Alors
Si la condition de transition a est vraie,	S_1_1 et S_1_2 sont exécutées.
Si les étapes S_1_1 et S_1_2 ont été activées,	les séquences sont exécutées indépendamment.
Si S_1_2 est active et que la condition de transition b est vraie,	la séquence passe de S_1_2 à S_1_3.
Si S_1_1 et S_1_3 sont actives et que la condition de transition c, de la transition commune est vraie,	S_1_1 et S_1_3 sont exécutées puis sautent vers S_1_1.
Si S_1_1 peut être activée par le saut,	seule la branche de S_1_1 devient active. La branche de S_1_2 ne devient pas active.
S_1_1 et S_1_3 ne devenant pas actives en même temps, la séquence ne peut pas continuer. Elle reste bloquée. Ce blocage peut être levé p. ex. par un deuxième jeton décalé dans le temps qui réactive l'étape S_1_2.	

Saut hors d'une séquence en ET

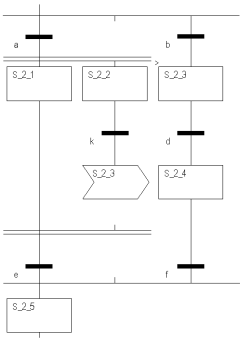
Présentation

Pour les jetons multiples, vous pouvez valider au choix le saut dans une séquence en ET ou hors d'une séquence en ET.

Dans tous les cas, des jetons supplémentaires sont générés.

Saut hors d'une séquence en ET.

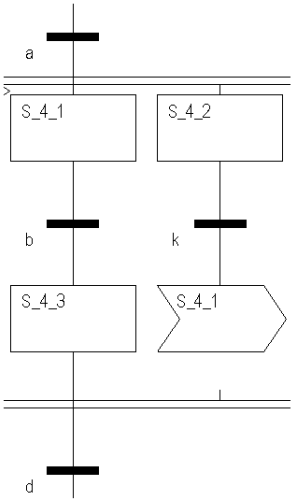
Saut hors d'une séquence en ET :



Si...		**Alors**	
Si la condition de transition `a` est vraie et que `b` est fausse,		`S_2_1` et `S_2_2` sont exécutées.	
Si les étapes `S_2_1` et `S_2_2` ont été activées,		les séquences sont exécutées indépendamment.	
Si la condition de transition `c` est vraie,		un saut est exécuté vers `S_2_3`.	
Le saut depuis la séquence en ET génère un deuxième jeton. Les deux jetons opèrent maintenant en parallèle, c.-à-d. `S_2_1` et `S_2_3` sont actives simultanément.			
Jeton 1 (S_2_1)		**Jeton 2 (S_2_3)**	
Si...	**Alors**	**Si...**	**Alors**
Si la condition de transition `e` est vraie,	la séquence passe à `S_2_5`.	Si la condition de transition `d` est vraie,	la séquence passe à `S_2_4`.
		Si la condition de transition `f` est vraie,	la séquence passe à `S_2_5`.
Si `S_2_5` est encore active en raison de l'activation de la condition de transition `e` (jeton 1), le jeton 2 s'arrête et la séquence est exécutée à nouveau en jeton unique. Si `S_2_5` n'est plus active (jeton 1), elle est réactivée par le jeton 2 et les deux jetons continuent à être exécutés en parallèle (jetons multiples).			

Saut entre deux branches d'une séquence en ET.

Saut entre deux branches d'une séquence en ET.



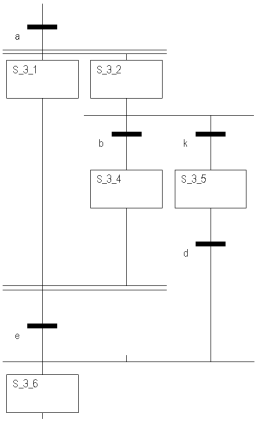
Si...	**Alors**
Si la condition de transition `a` est vraie,	la séquence passe par `S_4_1` et `S_4_2`.
Si les étapes `S_4_1` et `S_4_2` sont activées,	les séquences sont exécutées indépendamment.
Si la condition de transition `b` est vraie,	`S_4_3` est exécutée.
Si la condition de transition `c` est vraie,	un saut est exécuté vers `S_4_1`.
Le saut depuis une branche d'une séquence en ET génère un deuxième jeton. Les deux jetons opèrent maintenant en parallèle, c.à-d. `S_4_3` et `S_4_1` sont maintenant actives en même temps.	

Jeton 1 (S_4_3)		**Jeton 2 (S_4_1)**	
Si...	**Alors**	**Si...**	**Alors**
l'étape `S_4_3` est exécutée		l'étape `S_4_1` est exécutée	
		Si la condition de transition `b` est vraie,	`S_4_3` est exécutée.

Si lors de l'activation par le jeton 2, l'étape `S_4_3` est encore active (jeton 1), le jeton 2 s'arrête et la séquence est exécutée à nouveau en jeton unique.
Si lors de l'activation par le jeton 2, l'étape `S_4_3` n'est plus active (jeton 1), elle est réactivée par le jeton 2 et les deux jetons continuent à être exécutés en parallèle (jetons multiples).
Dans les deux cas, vous quittez la séquence en ET si la condition de transition `d` est vraie.

Quitter une séquence en ET par une divergence en OU

Quitter une séquence en ET par une divergence en OU



Si...		**Alors**	
Si la condition de transition a est vraie,		S_3_1 et S_3_2 sont exécutées.	
Si les étapes S_3_1 et S_3_2 sont activées,		les séquences sont exécutées indépendamment.	
Si la condition de transition b est fausse et que c est vraie,		la séquence passe à S_3_5.	
L'exécution depuis la séquence en ET via la divergence en OU génère un deuxième jeton. Les deux jetons fonctionnent maintenant en parallèle, c.-à-d. S_3_1 et S_3_5 sont actives simultanément.			
Jeton 1 (S_3_1)		**Jeton 2 (S_3_5)**	
Si...	**Alors**	**Si...**	**Alors**
S_3_4 ne pouvant devenir active, S_3_1 (jeton 1) reste active.		Si la condition de transition d est vraie,	la séquence passe à S_3_6.
Si la condition de transition a est vraie, S_3_1 et S_3_2 sont exécutées. Le jeton 2 est alors arrêté et la séquence est à nouveau exécutée comme jeton unique.			
Si la condition de transition a est vraie, S_3_1 et S_3_2 sont exécutées.			
		Si la condition de transition b est vraie et que c est fausse,	S_3_4 est exécutée.
S_3_4 ne pouvant devenir active, S_3_1 (jeton 1) reste active jusqu'à ce que la séquence par S_3_2 (jeton 2) et la transition b soit exécutée. Si S_4_4 n'est plus active (jeton 1), elle est réactivée par le jeton 2 et les deux jetons continuent à être exécutés en parallèle (jetons multiples). (La convergence des deux jetons peut déjà être réalisée dans S_4_3).			

Liste d'instructions (IL)

14

Objet de ce sous-chapitre

Ce chapitre décrit le langage de programmation Liste d'instructions IL conforme à la norme CEI 61131.

Contenu de ce chapitre

Ce chapitre contient les sous-chapitres suivants :

14.1 Remarques générales sur le langage liste d’instructions IL

Objet de ce sous-chapitre

Ce sous-chapitre vous donne un aperçu général du langage liste d’instructions IL.

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Informations générales sur la liste d'instructions IL

Présentation

Le langage de programmation Liste d'instructions (IL) vous permet par exemple d'appeler des fonctions et des blocs fonction de façon conditionnelle ou inconditionnelle, de lancer des affectations et d'exécuter des sauts conditionnels ou inconditionnels au sein de la section.

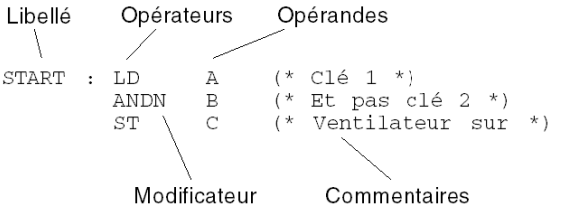
Instructions

Une liste d'instructions se compose d'une chaîne d'instructions.

Chaque instruction commence dans une nouvelle ligne et se compose :
- d'un opérateur *(voir page 469)*,
- le cas échéant, d'un modificateur *(voir page 467)* et,
- si nécessaire un ou plusieurs opérandes *(voir page 464)*

Si plusieurs opérandes sont utilisés, ceux-ci sont séparés par des virgules. Il est possible qu'un libellé *(voir page 480)* soit au début de l'instruction. Ce libellé est suivi par deux points. Un commentaire *(voir page 482)* peut suivre l'instruction.

Exemple :



Structure du langage de programmation

IL est un langage de programmation dit orienté accumulateur, ce qui signifie que chaque instruction utilise ou modifie le contenu de l'accumulateur (une sorte de mémoire temporaire). CEI 61131 désigne cet accumulateur comme "résultat".

Pour cette raison, une liste d'instructions doit toujours commencer par l'opérande `LD` ("Charger en commande accumulateur").

Exemple d'addition :

Commande	Signification
`LD 10`	La valeur 10 chargée dans l'accumulateur.
`ADD 25`	Le contenu de l'accumulateur est additionné à 25.
`ST A`	Le résultat est stocké dans la variable `A`. Le contenu de la variable `A` et de l'accumulateur est désormais 35. Toute nouvelle instruction fonctionnera avec le contenu de l'accumulateur "35" si elle ne commence pas avec `LD`.

Les opérations de comparaison se rapportent aussi toujours à l'accumulateur. Le résultat booléen de la comparaison est stocké dans l'accumulateur et devient ainsi son contenu actuel.

Exemple de comparaison :

Commande	Signification
`LD B`	La valeur `B` est chargée dans l'accumulateur.
`GT 10`	Le contenu de l'accumulateur est comparé à 10.
`ST A`	Le résultat de la comparaison est stocké dans la variable `A`. Si `B` est inférieur ou égal à 10, la valeur de la variable `A` et du contenu de l'accumulateur est 0 (FALSE). Si `B` est supérieur à 10, la valeur de la variable `A` et du contenu de l'accumulateur est 1 (TRUE).

Taille de la section

La taille d'une ligne d'instruction est limitée à 300 caractères.

La longueur d'une section IL n'est pas limitée au sein de l'environnement de programmation. La longueur d'une section IL n'est limitée que par la taille de la mémoire de l'automate.

Syntaxe

Les identificateurs et les mots-clés ne sont pas sensibles à la casse.

Les caractères d'espacement et de tabulation n'ont aucune influence sur la syntaxe et ils peuvent être utilisés librement.

Exception : Les caractères d'espacement et de tabulation ne sont pas autorisés dans :
- les mots-clés,
- les valeurs littérales,
- les valeurs,
- les identificateurs,
- les variables et
- combinaisons du limiteur [par ex., `(*` pour commentaires)].

Ordre d'exécution

L'exécution des instructions s'effectue ligne par ligne, du haut vers le bas. Cette suite de caractères peut être modifiée par mise entre parenthèses.

Si, par exemple, `A`, `B`, `C` et `D` ont les valeurs 1, 2, 3 et 4, et sont calculées comme suit :

```
LD A
ADD B
SUB C
MUL C
ST E
```

le résultat dans `E` sera 0.

Pour un calcul tel que :

```
LD A
ADD B
SUB(
LD C
MUL D
)
ST E
```

le résultat dans `E` sera -9.

Comportement en cas d'erreur

Les conditions suivantes seront traitées comme des erreurs lors de l'exécution d'une expression, par exemple :
- tentative de division par 0.
- les opérandes n'ont pas le type de données correct pour l'opération.
- le résultat d'une opération numérique dépasse la plage de valeurs de son type de données.

Conformité CEI

Pour la description de la conformité CEI du langage IL, voir Conformité CEI *(voir page 657)*.

Opérandes

Présentation

Les opérateurs sont utilisés sur les opérandes.

Un opérande peut être :
- une adresse,
- un libellé,
- une variable,
- une variable multi-éléments,
- un élément d'une variable multi-éléments,
- une sortie EFB/DFB ou
- un appel d'EFB/de DFB.

Types de données

L'opérande et le contenu actuel de l'accumulateur doivent obligatoirement avoir le même type de données. Si des opérandes de différents types de données doivent être traités, une conversion de types doit obligatoirement être effectuée auparavant.

Dans l'exemple, la variable Integer `i1` est convertie en une variable Real, avant d'être ajoutée à la variable Real `r4`.

```
LD  i1
INT_TO_REAL
ADD r4
ST  r3
```

Comme exception à cette règle, des variables du type de données `TIME` peuvent être multipliées par des variables du type de données `INT`, `DINT`, `UINT` ou `UDINT` ou divisées par ces dernières.

Opérations autorisées :
- ```
  LD  timeVar1
  DIV dintVar1
  ST  timeVar2
  ```
- ```
  LD  timeVar1
  MUL intVar1
  ST  timeVar2
  ```
- ```
  LD  timeVar1
  MUL 10
  ST  timeVar2
  ```

Cette fonction est considérée comme " indésirable " par la norme CEI 61131-3.
  ```

Utilisation directe d'adresses

Les adresses peuvent être utilisées directement (sans déclaration préalable). Dans ce cas, le type de données est directement affecté à l'adresse. L'affectation a lieu via le "préfixe de taille".

Le tableau suivant indique les différents préfixes de taille.

Préfixe de taille / Symbole	Exemple	Type de données
pas de préfixe	`%I10`, `%CH203.MOD`, `%CH203.MOD.ERR`	`BOOL`
`X`	`%M`**`X`**`20`	`BOOL`
`B`	%**QB**102.3	`BYTE`
`W`	`%K`**`W`**`4`3	`INT`
`D`	`%Q`**`D`**`100`	`DINT`
`F`	`%M`**`F`**`100`	`REAL`

Utilisation d'autres types de données

Si d'autres types de données doivent être affectés en tant que types de données par défaut d'une adresse, cela doit faire l'objet d'une déclaration explicite. L'éditeur de variables facilite la déclaration de ces variables. Il n'est pas possible de déclarer directement le type de données d'une adresse dans une section ST (par exemple la déclaration `AT %MW1 : UINT` ; non permise).

Exemple : les variables ci-dessous sont déclarées dans l'éditeur de variables.

```
UnlocV1 : ARRAY [1..10] OF INT;
LocV1 :   ARRAY [1..10] OF INT AT %MW100;
LocV2 :   TIME AT %MW100;
```

Les appels ci-dessous sont donc corrects du point de vue de la syntaxe :

```
%MW200 := 5;
LD LocV1[%MW200]
ST UnlocV1[2]

LD t#3s
ST LocV2
```

Accès aux variables de champs

Lors d'un accès aux valeurs d'un tableau (ARRAY), seuls les libellés et les variables du type INT, DINT, UINT et UDINT sont autorisés pour l'index.

L'index d'un élément ARRAY peut être négatif si la limite inférieure de la plage est négative.

Exemple : enregistrement d'une valeur d'un tableau

```
LD var1[i]
ST var2.otto[4]
```

Modificateur

Présentation

Les modificateurs influencent l'exécution de l'opérateur (voir *Opérateurs, page 469*).

Tableau des modificateurs

Tableau des modificateurs :

Modificateur	Applicable sur les opérandes du type de données	Description
N	BOOL, BYTE, WORD, DWORD	Le modificateur N est utilisé pour inverser bit à bit la valeur d'un opérande. **Exemple :** Dans l'exemple C est égal à 1, si A est égal à 1 et B est égal à 0. LD A AND**N** B ST C
C	BOOL	Le modificateur C est utilisé pour exécuter l'instruction associée, si la valeur de l'accu est 1 (TRUE). **Exemple :** Dans l'exemple, le saut après START est realize uniquement lorsque A est égal à 1 (TRUE) et B à 1 (TRUE). LD A AND B JMP**C** START
CN	BOOL	Si le modificateur C est combiné avec le modificateur N, l'instruction associée est exécutée seulement si la valeur de l'accumulateur est un 0 booléen (FALSE). **Exemple :** Dans l'exemple, le saut vers START est exécuté seulement si A est 0 (FALSE) et B est 0 (FALSE). LD A AND B JMP**CN** START

Modificateur	Applicable sur les opérandes du type de données	Description

(	toutes	Le modificateur Parenthèse gauche (est utilisé pour repousser l'évaluation de l'opérande, jusqu'à ce que l'opérateur Parenthèse droite) apparaisse. Le nombre d'opérations Parenthèse droite doit être égal au nombre de modificateurs Parenthèse gauche. Il est possible d'imbriquer les parenthèses.
		Exemple : Dans l'exemple, E a pour valeur 1, si C et/ou D sont définis sur 1 et si A et B ont aussi la valeur 1.
		`LD A`
		`AND B`
		`AND( C`
		`OR D`
		`)`
		`ST E`
		L'exemple peut être programmé de la façon suivante :
		`LD A`
		`AND B`
		`AND(`
		`LD C`
		`OR D`
		`)`
		`ST E`

Opérateurs

Introduction

Un opérateur est un symbole pour :

- une opération arithmétique à exécuter,
- une opération logique à exécuter ou
- l'appel d'un bloc fonction élémentaire, d'un DFB ou d'un sous-programme.

Les opérateurs sont génériques, ce qui signifie qu'ils s'adaptent automatiquement au type de données de l'opérande.

Opérateurs de chargement et d'enregistrement

Opérateurs de chargement et d'enregistrement du langage IL :

Opérateur	Modificateur	Signification	Opérandes	Description
LD	N (uniquement pour les opérandes du type de données BOOL, BYTE, WORD ou DWORD)	Charge la valeur de l'opérande dans l'accumulateur	Valeur littérale, variable, adresse directe avec type de données quelconque	Avec LD, la valeur d'un opérande est chargée dans l'accumulateur. Les données de l'accumulateur s'adaptent automatiquement au type de données de l'opérande. Cela s'applique également aux types de données dérivés. **Exemple :** Dans l'exemple donné, la valeur de A est chargée dans l'accumulateur, ajoutée à la valeur de B, puis le résultat est enregistré dans E. **LD** A ADD B ST E
ST	N (uniquement pour les opérandes du type de données BOOL, BYTE, WORD ou DWORD)	Enregistre la valeur de l'accumulateur dans l'opérande.	Variable, adresse directe avec type de données au choix	Avec ST, la valeur actuelle de l'accumulateur est enregistrée dans l'opérande. Le type de données de l'opérande doit correspondre au type de données de l'accumulateur. **Exemple :** Dans l'exemple donné, la valeur de A est chargée dans l'accumulateur, ajoutée à la valeur de B, puis le résultat est enregistré dans E. LD A ADD B **ST** E Le fait que ST soit suivi d'un LD ou pas détermine si l'on continue à travailler avec le résultat antérieur. **Exemple :** Dans l'exemple donné, la valeur de A est chargée dans l'accumulateur, ajoutée à la valeur de B, puis le résultat est enregistré dans E. Ensuite la valeur de B est déduite de la valeur de E (contenu actuel de l'accumulateur) et le résultat est enregistré dans C. LD A ADD B **ST** E SUB 3 **ST** C

Opérateurs de paramétrage et de réinitialisation

Opérateurs de paramétrage et de réinitialisation du langage IL

Opérateur	Modificateur	Signification	Opérandes	Description
S	-	Définit l'opérande sur 1 si le contenu de l'accumulateur est 1.	Variable, adresse directe du type de données BOOL	S permet de définir l'opérande sur 1 lorsque le contenu de l'accumulateur actuel est un 1 booléen. **Exemple :** Dans cet exemple, la valeur de A est chargée dans l'accumulateur. Si le contenu de l'accumulateur (valeur de A) est 1, alors OUT est défini sur 1. LD A **S** OUT Cet opérande est habituellement utilisé avec l'opérateur de réinitialisation R. **Exemple :** L'exemple montre un retournement RS (réinitialisation dominante) commandé via les deux variables booléennes A et C. LD A **S** OUT LD C R OUT
R	-	Définit l'opérande sur 0 si le contenu de l'accumulateur est 1.	Variable, adresse directe du type de données BOOL	R permet de définir l'opérande sur 0 lorsque le contenu actuel de l'accumulateur est un 1 booléen. **Exemple :** Dans cet exemple, la valeur de A est chargée dans l'accumulateur. Si le contenu de l'accumulateur (valeur de A) est 1, alors OUT est défini sur 0. LD A **R** OUT Cet opérande est habituellement utilisé avec l'opérateur de paramétrage S. **Exemple :** L'exemple montre un retournement SR (paramétrage dominant) commandé via les deux variables booléennes A et C. LD A **R** OUT LD C S OUT

Opérateurs logiques

Opérateurs logiques du langage IL :

Opérateur	Modificateur	Signification	Opérandes	Description
AND	N, N(, (	ET logique	Valeur littérale, variable, adresse directe du type de données BOOL, BYTE, WORD ou DWORD	L'opérateur AND établit une liaison ET logique entre le contenu de l'accumulateur et l'opérande. Pour les types de données BYTE, WORD et DWORD, la liaison est effectuée par bit. **Exemple :** Dans l'exemple donné, D aura pour valeur 1, si A, B et C sont sur 1. LD A **AND** B **AND** C ST D
OR	N, N(, (	OU logique	Valeur littérale, variable, adresse directe du type de données BOOL, BYTE, WORD ou DWORD	L'opérateur OR établit une liaison OU logique entre le contenu de l'accumulateur et l'opérande. Pour les types de données BYTE, WORD et DWORD, la liaison est effectuée par bit. **Exemple :** Dans l'exemple donné, D aura pour valeur 1, si A ouB est sur 1 et si C est sur 1. LD A **OR** B **OR** C ST D

Opérateur	Modificateur	Signification	Opérandes	Description
XOR	N, N(, (	OU exclusif logique	Valeur littérale, variable, adresse directe du type de données BOOL, BYTE, WORD ou DWORD	L'opérateur XOR établit une liaison OU exclusif logique entre le contenu de l'accumulateur et l'opérande. Si plus de deux opérandes sont reliés, le résultat de l'opération est à l'état 1 pour un nombre impair d'états 1 et à l'état 0 pour un nombre pair d'états 1. Pour les types de données BYTE, WORD et DWORD, la liaison est effectuée par bit. **Exemple :** Dans l'exemple, D est égal à 1 si A ouB est défini sur 1. Si A et B ont le même état (tous deux 0 ou 1), D est égal à 0. LD A **XOR** B ST D Si plus de deux opérandes sont reliés, le résultat de l'opération est à l'état 1 pour un nombre impair d'états 1 et à l'état 0 pour un nombre pair d'états 1. **Exemple :** Dans l'exemple, F a pour valeur 1 si 1 ou 3 opérandes sont définis sur 1. F a pour valeur 0 si 0, 2 ou 4 opérandes sont définis sur 1. LD A **XOR** B **XOR** C **XOR** D **XOR** E ST F
NOT	-	Négation logique (complément)	Contenu d'accumulateur du type de données BOOL, BYTE, WORD ou DWORD	NOT permet d'inverser le contenu de l'accumulateur par bit. **Exemple :** Dans l'exemple donné, B aura pour valeur 1, si A est sur 0 et B a pour valeur 0, si A est sur 1. LD A **NOT** ST B

Opérateurs arithmétiques

Opérateurs arithmétiques du langage IL :

Opérateur	Modificateur	Signification	Opérandes	Description
ADD	(	Addition	Valeur littérale, variable, adresse directe du type de données INT, DINT, UINT, UDINT, REAL ou TIME	ADD permet d'ajouter la valeur de l'opérande à la valeur du contenu de l'accumulateur. **Exemple :** L'exemple correspond à la formule D = A + B + C LD A **ADD** B **ADD** C ST D
SUB	(	Soustraction	Valeur littérale, variable, adresse directe du type de données INT, DINT, UINT, UDINT, REAL ou TIME	SUB permet de retirer la valeur de l'opérande du contenu de l'accumulateur. **Exemple :** L'exemple correspond à la formule D = A - B - C LD A **SUB** B **SUB** C ST D
MUL	(	Multiplication	Valeur littérale, variable, adresse directe du type de données INT, DINT, UINT, UDINT ou REAL	MUL permet de multiplier le contenu de l'accumulateur par la valeur de l'opérande. **Exemple :** L'exemple correspond à la formule D = A * B * C LD A **MUL** B **MUL** C ST D **Remarque :** La fonction MULTIME de la bibliothèque obsolète est destinée aux multiplications du type de données Time.
DIV	(	Division	Valeur littérale, variable, adresse directe du type de données INT, DINT, UINT, UDINT ou REAL	DIV permet de diviser le contenu de l'accumulateur par la valeur de l'opérande. **Exemple :** L'exemple correspond à la formule D = A / B / C LD A **DIV** B **DIV** C ST D **Remarque :** La fonction DIVTIME de la bibliothèque obsolète est destinée aux divisions du type de données Time.

Opérateur	Modificateur	Signification	Opérandes	Description

MOD	(	Division modulo	Valeur littérale, variable, adresse directe du type de données INT, DINT, UINT ou UDINT	MOD permet de diviser la valeur du premier opérande par la valeur du deuxième et de sortir le reste de la division (modulo) comme résultat.
				Exemple : Dans l'exemple donné,
				• C aura pour valeur 1, si A est égal à 7 et B à 2
				• C aura pour valeur 1, si A est égal à 7 et B à -2
				• C aura pour valeur -1, si A est égal à -7 et B à 2
				• C aura pour valeur -1, si A est égal à -7 et B à -2
				`LD A`
				`MOD B`
				`ST C`

Opérateurs de comparaison

Opérateurs de comparaison du langage IL :

Opérateur	Modificateur	Signification	Opérandes	Description

GT	(	Comparaison : >	Valeur littérale, variable, adresse directe du type de données BOOL, BYTE, WORD, DWORD, STRING, INT, DINT, UINT, UDINT, REAL, TIME, DATE, DT ou TOD	GT permet de comparer le contenu de l'accumulateur au contenu de l'opérande. Si le contenu de l'accumulateur est supérieur au contenu de l'opérande, le résultat est un 1 booléen. Si le contenu de l'accumulateur est inférieur ou égal au contenu de l'opérande, le résultat est un 0 booléen.
				Exemple : Dans l'exemple donné, la valeur de D est 1, si D est supérieur à 10. Dans le cas contraire, la valeur de D est 0.
				`LD A`
				`GT 10`
				`ST D`

GE	(	Comparaison : >=	Valeur littérale, variable, adresse directe du type de données BOOL, BYTE, WORD, DWORD, STRING, INT, DINT, UINT, UDINT, REAL, TIME, DATE, DT ou TOD	GE permet de comparer le contenu de l'accumulateur au contenu de l'opérande. Si le contenu de l'accumulateur est supérieur/égal au contenu de l'opérande, le résultat est un 1 booléen. Si le contenu de l'accumulateur est inférieur au contenu de l'opérande, le résultat est un 0 booléen.
				Exemple : Dans l'exemple donné, la valeur de D est 1, si D est supérieur ou égal à 10. Dans le cas contraire, la valeur de D est 0.
				`LD A`
				`GE 10`
				`ST D`

Opérateur	Modificateur	Signification	Opérandes	Description
EQ	(	Comparaison : =	Valeur littérale, variable, adresse directe du type de données BOOL, BYTE, WORD, DWORD, STRING, INT, DINT, UINT, UDINT, REAL, TIME, DATE, DT ou TOD	EQ permet de comparer le contenu de l'accumulateur au contenu de l'opérande. Si le contenu de l'accumulateur est égal à celui de l'opérande, le résultat est un 1 booléen. Si le contenu de l'accumulateur n'est pas égal à celui de l'opérande, le résultat est un 0 booléen. **Exemple :** Dans l'exemple donné, la valeur de D est 1, si A est égal à 10. Dans le cas contraire, la valeur de D est 0. LD A **EQ** 10 ST D
NE	(	Comparaison : <>	Valeur littérale, variable, adresse directe du type de données BOOL, BYTE, WORD, DWORD, STRING, INT, DINT, UINT, UDINT, REAL, TIME, DATE, DT ou TOD	NE permet de comparer le contenu de l'accumulateur au contenu de l'opérande. Si le contenu de l'accumulateur n'est pas égal à celui de l'opérande, le résultat est un 1 booléen. Si le contenu de l'accumulateur est égal à celui de l'opérande, le résultat est un 0 booléen. **Exemple :** Dans l'exemple donné, la valeur de D est 1, si A n'est pas égal à 10. Dans le cas contraire, la valeur de D est 0. LD A **NE** 10 ST D
LE	(	Comparaison : <=	Valeur littérale, variable, adresse directe du type de données BOOL, BYTE, WORD, DWORD, STRING, INT, DINT, UINT, UDINT, REAL, TIME, DATE, DT ou TOD	LE permet de comparer le contenu de l'accumulateur au contenu de l'opérande. Si le contenu de l'accumulateur est inférieur ou égal à celui de l'opérande, le résultat est un 1 booléen. Si le contenu de l'accumulateur est supérieur à celui de l'opérande, le résultat est un 0 booléen. **Exemple :** Dans l'exemple donné, la valeur de D est 1, si D est inférieur ou égal à 10. Dans le cas contraire, la valeur de D est 0. LD A **LE** 10 ST D

Opérateur	Modificateur	Signification	Opérandes	Description
LT	(	Comparaison : <	Valeur littérale, variable, adresse directe du type de données BOOL, BYTE, WORD, DWORD, STRING, INT, DINT, UINT, UDINT, REAL, TIME, DATE, DT ou TOD	LT permet de comparer le contenu de l'accumulateur au contenu de l'opérande. Si le contenu de l'accumulateur est inférieur à celui de l'opérande, le résultat est un 1 booléen. Si le contenu de l'accumulateur est supérieur ou égal à celui de l'opérande, le résultat est un 0 booléen. **Exemple :** Dans l'exemple donné, la valeur de D est 1, si D est inférieur à 10. Dans le cas contraire, la valeur de D est 0. LD A **LT** 10 ST D

Opérateurs d'appel

Opérateurs d'appel du langage IL :

Opérateur	Modificateur	Signification	Opérandes	Description
CAL	C, CN (uniquement si le contenu de l'accumulateur est du type de données BOOL)	Appel d'un bloc fonction, d'un DFB ou d'un sous-programme	Nom d'instance d'un bloc fonction, d'un DFB ou d'un sous-programme	CAL permet d'appeler un bloc fonction, un DFB ou un sous-programme avec ou sans conditions. Voir également *Appel de blocs fonction élémentaires et de blocs fonction dérivés, page 489* et *Appel de sous-programme, page 479*
NOM_DE_FONCTION	-	Exécution d'une fonction	Valeur littérale, variable, adresse directe (le type de données dépend de la fonction)	Le nom de fonction vous permet d'exécuter une fonction. Voir également *Appel de fonctions élémentaires, page 484*
NOM_DE_PROCEDURE	-	Exécution d'une procédure	Valeur littérale, variable, adresse directe (le type de données dépend de la procédure)	Le nom de procédure vous permet d'exécuter une procédure. Voir également *Procédures d'appel, page 500*

Opérateurs de structuration

Opérateurs de structuration du langage IL :

Opérateur	Modificateur	Signification	Opérandes	Description
JMP	C, CN (uniquement si le contenu de l'accumulateur est du type de données BOOL)	Saut vers l'étiquette	ETIQUETTE	JMP permet d'effectuer un saut avec ou sans conditions vers une étiquette. Voir également *Libellés et sauts, page 480*
RET	C, CN (uniquement si le contenu de l'accumulateur est du type de données BOOL)	Retour dans l'unité organisationnelle supérieure suivante du programme	-	Des opérateurs RETURN peuvent être utilisés dans des blocs de fonction dérivés DFB et des SR (sous-programmes). Des opérateurs RETURN ne peuvent pas être utilisés dans le programme principal. ● Dans un DFB, un opérateur RETURN force le retour au programme qui a appelé le DFB. ● Le reste de la section de DFB contenant l'opérateur RETURN n'est pas exécuté. ● Les sections suivantes du DFB ne sont pas exécutées. Le programme qui a appelé le DFB sera exécuté après retour du DFB. Si le DFB est appelé par un autre DFB, le DFB appelant sera exécuté après le retour. ● Dans un SR, un opérateur RETURN force le retour au programme qui a appelé le SR. ● Le reste du SR contenant l'opérateur RETURN n'est pas exécuté. Le programme qui a appelé le SR sera exécuté après retour du SR.
)	-	Traitement d'opérations placées en attente	-	La parenthèse droite) permet de lancer l'édition de l'opérateur en attente. Le nombre d'opérations Parenthèse droite doit être égal au nombre de modificateurs Parenthèse gauche. Il est possible d'imbriquer les parenthèses. **Exemple :** Dans l'exemple donné, E aura pour valeur 1, si C et/ou D est sur 1 et si A et B sont sur 1. LD A AND B AND(C OR D) ST E

Appel de sous-programme

Appeler un sous-programme

En IL, l'appel d'un sous-programme se compose de l'opérateur `CAL`, suivi du nom de la section de sous-programme, puis d'une liste de paramètres vide (facultative).

Les appels de sous-programmes ne fournissent pas de valeurs de retour.

Le sous-programme à appeler doit se trouver dans la même tâche que la section IL appelante.

Il est possible d'appeler des sous-programmes au sein de sous-programmes.

par exemple:

```
ST A
CAL SubroutineName ()
LD B
```

ou

```
ST A
CAL SubroutineName
LD B
```

Les appels de sous-programmes sont un complément de la norme CEI 61131-3 et doivent être activés de manière explicite.

Dans les sections d'actions SFC, les appels de sous-programmes ne sont autorisés que si le mode Multitoken a été activé.

Libellés et sauts

Présentation

Les libellés servent de cible à atteindre pour les sauts.

Propriétés des libellés :

Propriétés des étiquettes :

- les étiquettes doivent toujours être le premier élément d'une ligne.
- les étiquettes doivent être uniques dans toute la section, on ne fait pas de distinction ici entre majuscules et minuscules.
- la longueur maximale d'un étiquette est de 32 caractères.
- Les libellés doivent satisfaire aux conventions de désignation CEI.
- les étiquettes sont séparés par deux points `:` de la commande suivante.
- Les libellés doivent se trouver au début des "expressions" car uniquement des valeurs non-définies peuvent se trouver dans l'accumulateur.
  Exemple :

```
start: LD A
       AND B
       OR C
       ST D
       JMP start
```

Propriétés des sauts :

Propriétés des sauts

- L'opération `JMP` exécute un saut avec ou sans conditions vers un libellé.
- `JMP` peut être utilisé avec les modificateurs `C` et `CN` (uniquement quand le contenu d'accumulateur actuel est du type de données `BOOL`).
- Les sauts sont possibles au sein des sections de DFB et de programme.
- Les sauts ne sont possibles qu'au sein d'une même section.

Des destinations de saut possibles sont :

- la première instruction `LD` d'un appel d'EFB/de DFB avec affectation de paramètres d'entrée (voir `start2`),
- une instruction `LD` "normale" (voir `start1`),
- une instruction `CAL` ne travaillant pas avec l'affectation de paramètres d'entrée (voir `start3`),
- une instruction `JMP` (voir `start4`),
- la fin d'une liste d'instructions (voir `start5`).

Exemple

```
start2: LD A
        ST counter.CU
        LD B
        ST counter.R
        LD C
        ST counter.PV
        CAL counter
        JMPCN start4
start1: LD A
        AND B
        OR C
        ST D
        JMPC start3
        LD A
        ADD E
        JMP start5
start3: CAL counter (
        CU:=A
        R:=B
        PV:=C )
        JMP start1
        LD A
        OR B
        OR C
        ST D
start4: JMPC start1
        LD C
        OR B
start5: ST A
```

Commentaire

Description

Dans l'éditeur IL, les commentaires commencent par la chaîne de caractères `(*` et se terminent par la chaîne de caractères `*)`. Vous pouvez entrer un commentaire quelconque entre ces deux chaînes de caractères.

Selon la norme CEI 61131-3, il n'est pas possible d'imbriquer des commentaires. Toutefois, si des commentaires sont imbriqués, ils doivent être activés de manière explicite.

14.2 Appel de fonctions élémentaires, de blocs fonction élémentaires, de blocs fonction dérivés et de procédures

Objet de ce sous-chapitre

Appel de fonctions élémentaires, de blocs fonction élémentaires, de blocs fonction dérivés et de procédures dans le langage de programmation IL.

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Appel de fonctions élémentaires

Utilisation des fonctions

Les fonctions élémentaires sont disponibles sous forme de bibliothèques. La logique des fonctions est créée dans le langage de programmation C et ne peut pas être modifiée dans l'éditeur IL.

Les fonctions n'ont pas d'état interne. Lorsque les valeurs d'entrée sont identiques, la valeur de sortie est la même à chaque exécution de la fonction. Par exemple, l'addition de deux valeurs donne toujours le même résultat. Pour certaines fonctions élémentaires, il est possible d'augmenter le nombre d'entrées.

Les fonctions élémentaires n'ont qu'une seule valeur de renvoi (sortie).

Paramètres

Pour importer des valeurs dans une fonction ou exporter des valeurs d'une fonction, on a besoin d'« entrées » et d'une « sortie ». Ces entrées et cette sortie sont appelées « paramètres formels ».

Les états actuels du processus sont transmis aux paramètres formels. Ce sont les paramètres réels.

On peut utiliser un paramètre réel d'entrée de fonction de type :
- variable,
- adresse,
- valeur littérale.

On peut utiliser un paramètre réel de sortie de fonction de type :
- variable,
- adresse.

Le type des données du paramètre réel doit correspondre au type des données du paramètre formel. La seule exception concerne les paramètres formels génériques dont le type de données est déterminé par le paramètre réel.

De plus, pour les paramètres formels génériques du type de données `ANY_BIT`, des paramètres réels du type de données `INT` ou `DINT` (pas `UINT` ni `UDINT`) peuvent être utilisés.

Il s'agit d'une extension de la norme CEI 61131-3, qui doit être activée de manière explicite.

Exemple :

Autorisé :

```
AND (AnyBitParam := IntVar1, AnyBitParam2 := IntVar2)
```

Non autorisé :

```
AND_WORD (WordParam1 := IntVar1, WordParam2 := IntVar2)
```

(Dans ce cas `AND_INT` doit être utilisé.)

```
AND_ARRAY_WORD (ArrayInt, ...)
```

(Dans ce cas, un changement de type explicite doit être effectué via `INT_ARR_TO_WORD_ARR (...)`.

Pour l'appel formel, il n'est en principe pas nécessaire d'affecter une valeur à tous les paramètres formels. Pour connaître les types de paramètre formel pour lesquels cela est cependant impératif, veuillez vous reporter au tableau.

Type de paramètre	EDT	STRING	ARRAY	ANY_ARRAY	IODDT	STRUCT	FB	ANY
Entrée	-	-	+	+	+	+	+	+
VAR_IN_OUT	+	+	+	+	+	+	/	+
Sortie	-	-	-	-	-	-	/	-
+ paramètre réel impératif								
- paramètre réel non impératif								
/ non applicable								

Si aucune valeur n'est affectée à un paramètre formel, la valeur initiale est utilisée lors de l'exécution de la fonction. Si aucune valeur initiale n'est définie, la valeur par défaut (0) est utilisée.

Remarques sur la programmation

Veuillez tenir compte des remarques qui suivent sur la programmation :

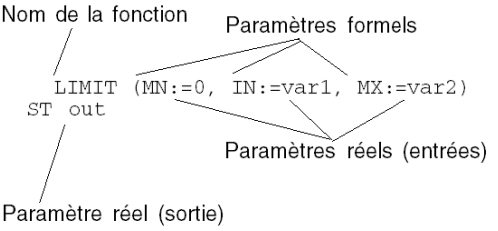
- Les fonctions ne sont exécutées que lorsque l'entrée `EN` = 1 ou lorsque l'entrée `EN` est désactivée (voir aussi EN and ENO *(voir page 488)*).
- Toutes les fonctions génériques sont surchargées. Cela signifie que les fonctions peuvent être appelées avec ou sans la saisie du type de données.
  Ex. :
  ```
  LD i1
  ADD i2
  ST i3
  ```
  est identique à
  ```
  LD i1
  ADD_INT i2
  ST i3
  ```
- Contrairement au langage ST, les fonctions ne peuvent pas être imbriquées dans le langage IL.
- Il existe deux façons d'appeler une fonction :
  - appel formel (appel d'une fonction avec les noms des paramètres formels),
  - appel informel (appel d'une fonction sans les noms des paramètres formels).

Appel formel

Avec ce type d'appel (avec les noms des paramètres formels), les fonctions sont appelées via une suite d'instructions qui comprend le nom de la fonction suivi d'une liste entre parenthèses des affectations de valeur (paramètres réels) aux paramètres formels. L'ordre d'énumération des paramètres formels **n'est pas important**. La liste des paramètres réels peut être éditée directement après une virgule. Après l'exécution de la fonction, le résultat est chargé dans l'accumulateur et peut être sauvegardé via `ST`.

Il est possible d'utiliser `EN` et `ENO` avec ce type d'appel.

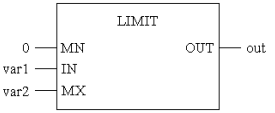
Appel d'une fonction avec les noms des paramètres formels :



ou

```
LIMIT ( MN:=0, IN:=var1, MX:=var2 ) ST out
```

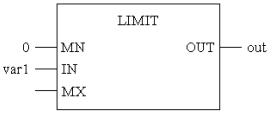
Appel de la même fonction dans FBD :



Lors d'un appel formel, il n'est pas nécessaire d'affecter une valeur à tous les paramètres formels (voir également Parameter *(voir page 484)*).

```
LIMIT (MN:=0, IN:=var1) ST out
```

Appel de la même fonction dans FBD :

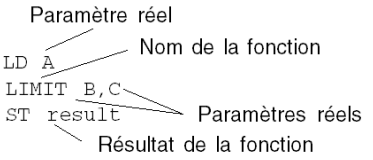


Appel informel

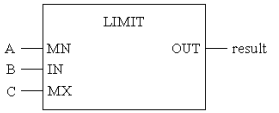
Avec ce type d'appel (sans les noms des paramètres formels), les fonctions sont appelées via une suite d'instructions qui comprend le chargement du premier paramètre réel dans l'accumulateur suivi du nom de la fonction, lui-même suivi d'une liste optionnelle des paramètres réels. L'ordre d'énumération des paramètres réels est **important**. La liste des paramètres réels ne peut pas être éditée. Après l'exécution de la fonction, le résultat est chargé dans l'accumulateur et peut être sauvegardé via `ST`.

`EN` et `ENO` ne peuvent **pas** être utilisés avec ce type d'appel.

Appel d'une fonction avec les noms des paramètres formels :



Appel de la même fonction dans FBD :



NOTE : veuillez noter que pour l'appel informel, la liste des paramètres réels ne doit **pas** être indiquée entre parenthèses. La norme CEI 61133-3 exige dans ce cas d'enlever les parenthèses, afin d'indiquer que le premier paramètre réel ne fait pas partie de la liste.

Appel informel **non valide** d'une fonction :

```
LD A
LIMIT (B,C)
```

Si la valeur à traiter (premier paramètre réel) se trouve déjà dans l'accumulateur, l'instruction de chargement n'est plus nécessaire.

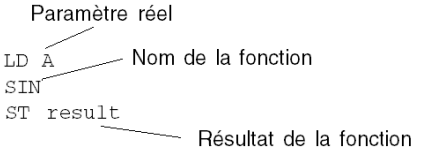
```
LIMIT B,C ST result
```

Si le résultat doit être directement utilisé, l'instruction de sauvegarde n'est plus nécessaire :

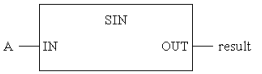
```
LD A LIMIT_REAL B,C MUL E
```

Si la fonction à exécuter n'a qu'une seule entrée, le nom de la fonction n'est pas suivi d'une liste de paramètres réels.

Appel d'une fonction avec un paramètre réel :



Appel de la même fonction dans FBD :



EN et ENO

Pour toutes les fonctions, une entrée EN et une sortie ENO peuvent être configurées.

Si la valeur d'EN est égale à « 0 », lorsque la fonction est appelée, les algorithmes définis par cette dernière ne sont pas exécutés et ENO est mis à « 0 ».

Si la valeur d'EN est égale à « 1 », lorsque la fonction est appelée, les algorithmes définis par la fonction sont exécutés. Une fois les algorithmes exécutés, la valeur de la sortie ENO est réglée sur « 1 ». Si une erreur se produit durant l'exécution de ces algorithmes, ENO est mis à « 0 ».

Si aucune valeur n'est attribuée à la broche EN à l'appel du FFB, l'algorithme défini par ce dernier est exécuté (comme lorsque EN a la valeur « 1 »).

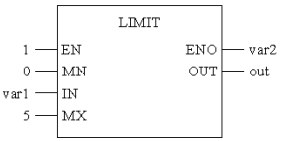
Si ENO est mis à « 0 » (en raison de EN = 0 ou d'une erreur d'exécution), la sortie de la fonction est mise à « 0 ».

Le comportement de sortie de la fonction ne dépend pas de l'appel de la fonction sans EN/ENO ou avec EN=1.

Si EN/ENO doivent être utilisés, l'appel de la fonction doit être exécuté comme un appel formel.

```
LIMIT (EN:=1, MN:=0, IN:=var1, MX:=5, ENO=>var2) ST out
```

Appel de la même fonction dans FBD :



Appel de blocs fonction élémentaires et de blocs fonction dérivés

Bloc fonction élémentaire

Les blocs fonction élémentaires ont des états internes. Pour des valeurs égales aux entrées, la valeur à la sortie peut être différente à chaque exécution du bloc fonction. Pour un compteur, par exemple, la valeur de la sortie est incrémentée.

Les blocs fonction peuvent comprendre plusieurs valeurs de renvoi (sorties).

Bloc fonction dérivé

Les blocs fonction dérivés (DFB) ont les mêmes caractéristiques que les blocs fonction élémentaires. Ils sont cependant créés par l'utilisateur dans les langages FBD, LD, IL et/ou ST.

Paramètres

Pour importer des valeurs dans un bloc fonction ou les exporter d'un bloc fonction, des entrées et des sorties sont nécessaires. Ces entrées et ces sorties sont appelées « paramètres formels ».

Les états actuels du processus sont transmis aux paramètres formels. Ce sont les « paramètres réels ».

Pour les entrées de bloc fonction, on peut utiliser un paramètre réel de type :
- variable,
- adresse,
- valeur littérale.

Pour les sorties de bloc fonction, on peut utiliser un paramètre réel de type :
- variable,
- adresse.

Le type des données du paramètre réel doit correspondre au type des données du paramètre formel. La seule exception concerne les paramètres formels génériques dont le type de données est déterminé par le paramètre réel.

Exception :

Pour les paramètres formels génériques de type de données `ANY_BIT`, des paramètres réels de type de données `INT` ou `DINT` (pas `UINT` ni `UDINT`) peuvent être utilisés.

Il s'agit d'une extension de la norme CEI 61131-3, qui doit être activée de manière explicite.

Exemple :

Autorisé :

```
AND (AnyBitParam := IntVar1, AnyBitParam2 := IntVar2)
```

Non autorisé :

```
AND_WORD (WordParam1 := IntVar1, WordParam2 := IntVar2)
```

(Dans ce cas `AND_INT` doit être utilisé.)

```
AND_ARRAY_WORD (ArrayInt, ...)
```

(Dans ce cas, un changement de type explicite doit être effectué via `INT_ARR_TO_WORD_ARR (...)`.

Il n'est en principe pas nécessaire d'affecter une valeur à tous les paramètres formels. Pour connaître les types de paramètre formel pour lesquels cela est cependant impératif, veuillez vous reporter au tableau.

Type de paramètre	EDT	STRING	ARRAY	ANY_ARRAY	IODDT	STRUCT	FB	ANY
EFB : Entrée	-	+	+	+	/	+	/	+
EFB : VAR_IN_OUT	+	+	+	+	+	+	/	+
EFB : Sortie	-	-	+	+	+	-	/	+
DFB : Entrée	-	+	+	+	/	+	/	+
DFB : VAR_IN_OUT	+	+	+	+	+	+	/	+
DFB : Sortie	-	-	+	/	/	-	/	+
+ paramètre réel impératif								
- paramètre réel non impératif								
/ non applicable								

Si aucune valeur n'est affectée à un paramètre formel, la valeur initiale est utilisée pendant l'exécution du bloc fonction. Si aucune valeur initiale n'est définie, la valeur par défaut (0) est utilisée.

Si aucune valeur n'est affectée à un paramètre formel et que le bloc fonction/DFB ait été instancié à plusieurs reprises, les instances appelées par la suite travaillent avec l'ancienne valeur.

Variables publiques

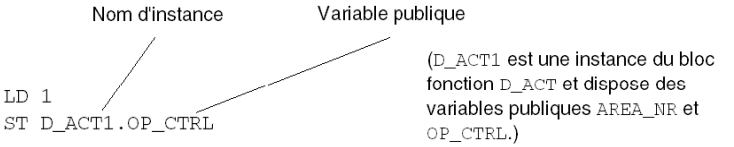
Certains blocs fonction disposent non seulement d'entrées et de sorties, mais également de variables publiques.

Ces variables permettent de transmettre des valeurs statistiques (valeurs non influencées par le procédé) au bloc fonction. Elles sont donc utilisées lors du paramétrage du bloc fonction.

Les variables publiques sont une extension de la norme CEI 61131-3.

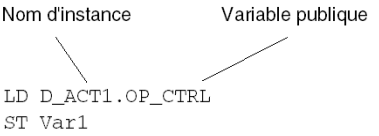
Des valeurs sont affectées aux variables publiques via leur valeur initiale ou par instructions de chargement et d'enregistrement.

Exemple :



Les valeurs des variables publiques sont ensuite lues à partir du nom d'instance du bloc fonction et du nom de la variable publique.

Exemple :



Variables privées

Certains blocs fonction disposent non seulement d'entrées, de sorties et de variables publiques, mais également de variables privées.

A l'instar des variables publiques, ces variables permettent de transmettre des valeurs statistiques (valeurs non influencées par le procédé) au bloc fonction.

Le programme utilisateur ne peut pas accéder à ces variables. Seule la table d'animation en a la capacité.

NOTE : les DFB imbriqués sont déclarés comme des variables privées du DFB parent. Ainsi, leurs variables ne sont pas accessibles via la programmation, mais via la table d'animation.

Les variables privées sont une extension de la norme CEI 61131-3.

Remarques sur la programmation

Veuillez tenir compte des remarques qui suivent sur la programmation :

- Les fonctions ne sont exécutées que lorsque l'entrée `EN` = 1 ou lorsque l'entrée `EN` est désactivée (voir aussi EN and ENO *(voir page 497)*).
- L'affectation de variables aux sorties de type `ANY` ou `ARRAY` doit être effectuée via l'opérateur `=>` (voir aussi Formal Form of CAL with a List of the Input Parameters *(voir page 492)*).
  Une affectation en dehors du cadre de l'appel de bloc fonction n'est pas possible. L'instruction

```
My_Var := My_SAH.OUT
```
n'est pas valide, la sortie `OUT` du bloc fonction `SAH` étant de type `ANY`.
L'instruction
```
Cal My_SAH (OUT=>My_Var)
```
est en revanche **valide**.

- Des conditions particulières s'appliquent lors de l'utilisation de variables VAR_IN_OUT *(voir page 498)*.
- L'utilisation des blocs fonction comprend deux parties :
  - la déclaration *(voir page 492)*
  - l'appel du bloc fonction
- Il existe quatre façons d'appeler un bloc fonction :
  - forme formelle de CAL avec liste des paramètres d'entrée *(voir page 492)* (appel avec les noms des paramètres formels)
    Des variables peuvent ainsi être affectées aux sorties via l'opérateur `=>`.
  - forme informelle de CAL avec liste des paramètres d'entrée *(voir page 494)* (appel sans les noms des paramètres formels)
  - CAL et chargement/sauvegarde *(voir page 495)* des paramètres d'entrée
  - usage des opérateurs d'entrée *(voir page 495)*
- Les instances de bloc fonction/DFB peuvent être appelées plusieurs fois, à l'exception des instances d'EFB de communication qui ne peuvent être appelées qu'une seule fois (voir Multiple Call of a Function Block Instance *(voir page 497)*).

Déclaration

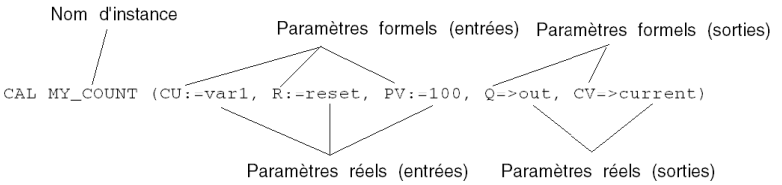
Avant d'être appelé, un bloc fonction doit être déclaré dans l'éditeur de variables.

Forme formelle de CAL avec liste des paramètres d'entrée

Avec cette forme d'appel (avec les noms des paramètres formels), les blocs fonction sont appelés via une instruction qui se compose de l'instruction `CAL`, suivie du nom d'instance du bloc fonction et d'une liste entre parenthèses des affectations de paramètres réels aux paramètres formels. L'affectation des paramètres formels des entrées s'effectue via l'affectation `:=` et l'affectation des paramètres formels des sorties via l'affectation `=>`. L'ordre d'énumération des paramètres formels d'entrées et de sorties **n'est pas important**. La liste des paramètres réels peut être éditée directement après une virgule.

Il est possible d'utiliser `EN` et `ENO` avec ce type d'appel.

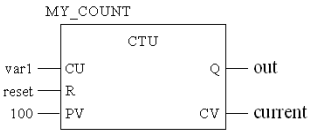
Appel d'un bloc fonction dans la forme formelle de CAL avec liste des paramètres d'entrée :



ou

```
CAL MY_COUNT (CU:=var1, R:=reset, PV:=100, Q=>out,
CV=>current)
```

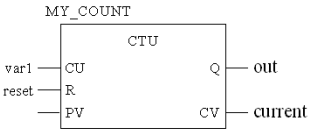
Appel du même bloc fonction dans FBD :



Il n'est pas nécessaire d'affecter une valeur à tous les paramètres formels (voir aussi Parameter *(voir page 489)*).

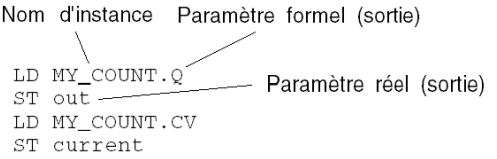
```
CAL MY_COUNT (CU:=var1, R:=reset, Q=>out, CV=>current)
```

Appel du même bloc fonction dans FBD :



Le déplacement de la valeur d'une sortie de bloc fonction peut également avoir lieu via le chargement de la sortie du bloc fonction (nom d'instance du bloc fonction séparé par un point du paramètre formel) suivi d'une sauvegarde.

Chargement et sauvegarde des sorties de bloc fonction :

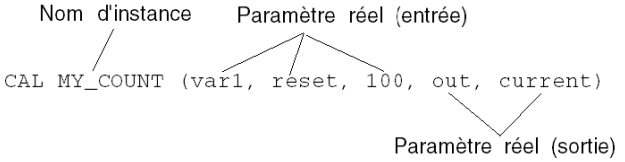


Forme informelle de CAL avec liste des paramètres d'entrée

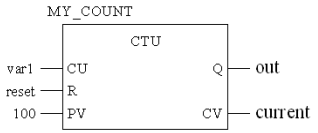
Avec cette forme d'appel (sans les noms des paramètres formels), les blocs fonction sont appelés via une instruction qui se compose de l'instruction `CAL`, suivie du nom d'instance du bloc fonction et d'une liste entre parenthèses des paramètres réels des entrées et sorties. L'ordre d'énumération des paramètres réels dans l'appel d'un bloc fonction **est important**. La liste des paramètres réels ne peut pas être éditée.

`EN` et `ENO` ne peuvent **pas** être utilisés avec ce type d'appel.

Appel d'un bloc fonction dans la forme informelle de CAL avec liste des paramètres d'entrée :



Appel du même bloc fonction dans FBD :



Même lors d'un appel informel, il n'est pas nécessaire d'affecter une valeur à tous les paramètres formels (voir également Parameter *(voir page 489)*).

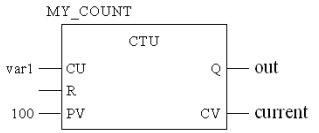
Il s'agit d'une extension de la norme CEI 61131-3, qui doit être activée de manière explicite.

Un champ de paramètre vide permet d'omettre un paramètre.

Appel avec un champ de paramètre vide :

```
CAL MY_COUNT (var1, , 100, out, current)
```

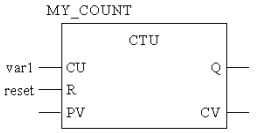
Appel du même bloc fonction dans FBD :



Si les paramètres formels sont omis à la fin, aucun champ de paramètre vide ne doit être utilisé.

```
MY_COUNT (var1, reset)
```

Appel du même bloc fonction dans FBD :



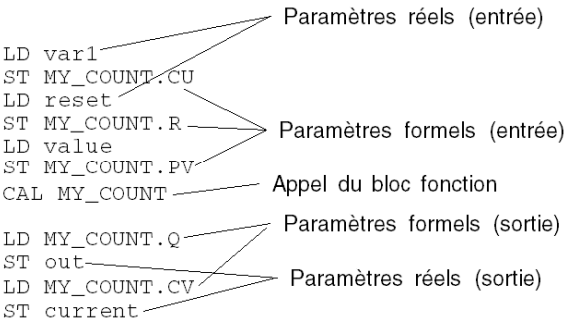
CAL et chargement/sauvegarde des paramètres d'entrée

Vous pouvez appeler les blocs fonction à l'aide d'une liste d'instructions composée du chargement des paramètres réels suivi de leur sauvegarde dans les paramètres formels, suivie de l'instruction `CAL`. L'ordre de chargement et de sauvegarde des paramètres **n'est pas important**.

Seules les instructions de chargement et de sauvegarde pour le bloc fonction en cours de paramétrage doivent figurer entre la première instruction de chargement du paramètre réel et l'appel du bloc fonction. Aucune autre instruction n'est autorisée à cet endroit.

Il n'est pas nécessaire d'affecter une valeur à tous les paramètres formels (voir aussi Parameter *(voir page 489)*).

CAL avec chargement/sauvegarde des paramètres d'entrée :



Usage des opérateurs d'entrée

Vous pouvez appeler les blocs fonction à l'aide d'une liste d'instructions composée du chargement des paramètres réels suivi de leur sauvegarde dans les paramètres formels, suivie d'un opérateur d'entrée. L'ordre de chargement et de sauvegarde des paramètres **n'est pas important**.

Seules les instructions de chargement et de sauvegarde pour le bloc fonction en cours de paramétrage doivent figurer entre la première instruction de chargement du paramètre réel et l'opérateur d'entrée du bloc fonction. Aucune autre instruction n'est autorisée à cet endroit.

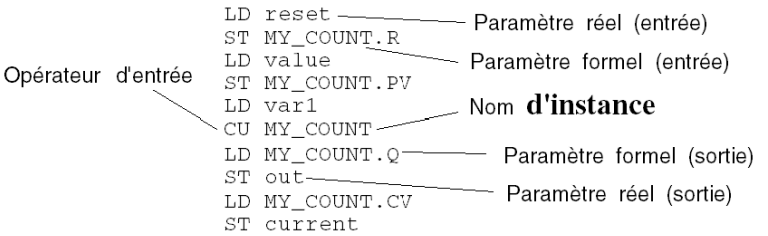
EN et ENO ne peuvent **pas** être utilisés avec ce type d'appel.

Il n'est pas nécessaire d'affecter une valeur à tous les paramètres formels (voir aussi Parameter *(voir page 489)*).

Reportez-vous au tableau pour connaître les opérateurs d'entrée disponibles pour les différents blocs fonction. Aucun autre opérateur d'entrée n'est disponible.

Opérateur d'entrée	Type FB
S1, R	SR
S, R1	RS
CLK	R_TRIG
CLK	F_TRIG
CU, R, PV	CTU_INT, CTU_DINT, CTU_UINT, CTU_UDINT
CD, LD, PV	CTD_INT, CTD_DINT, CTD_UINT, CTD_UDINT
CU, CD, R, LD, PV	CTUD_INT, CTUD_DINT, CTUD_UINT, CTUD_UDINT
IN, PT	TP
IN, PT	TON
IN, PT	TOF

Usage des opérateurs d'entrée :



Appel d'un bloc fonction sans entrées

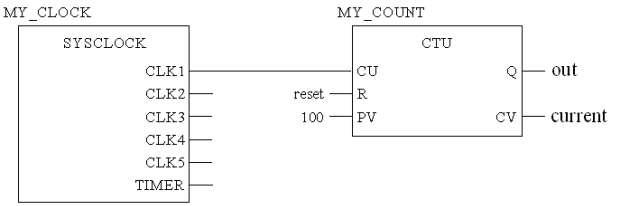
Même si le bloc fonction ne dispose pas d'entrées ou si les entrées ne sont pas à paramétrer, vous devez appeler le bloc fonction avant que ses sorties puissent être utilisées. Faute de quoi, le système transmettra les valeurs initiales des sorties, c'est-à-dire « 0 ».

Ex. :

Appel de blocs fonction dans IL :

```
CAL MY_CLOCK ()CAL MY_COUNT (CU:=MY_CLOCK.CLK1, R:=reset,
PV:=100) LD MY_COUNT.Q ST out LD MY_COUNT.CV ST current
```

Appel du même bloc fonction dans FBD :



Appel multiple d'une instance de bloc fonction

Les instances de bloc fonction/DFB peuvent être appelées plusieurs fois, à l'exception des instances d'EFB de communication qui ne peuvent être appelées qu'une seule fois.

L'appel multiple de la même instance de DFB/bloc fonction est par exemple utile dans les cas suivants :

- si le bloc fonction/DFB ne comporte aucune valeur interne ou si celle-ci n'est plus nécessaire pour un traitement ultérieur.
  Dans ce cas, l'appel multiple de la même instance de DFB/bloc fonction permet d'économiser de l'espace mémoire, car le code du bloc fonction/DFB n'est alors chargé qu'une seule fois.
  Le bloc fonction/DFB est pour ainsi dire traité comme une « fonction ».
- si le bloc fonction/DFB comprend une valeur interne, qui doit influencer différents endroits du programme (la valeur d'un compteur, par exemple, doit être augmentée dans différentes parties du programme).
  Dans ce cas, l'appel multiple de la même instance de bloc fonction/DFB permet d'économiser la mémoire des résultats intermédiaires pour un traitement ultérieur à un autre endroit du programme.

EN et ENO

Pour tous les blocs fonction/DFB, une entrée EN et une sortie ENO peuvent être configurées.

Au cas où la valeur d'EN est égale à « 0 », lorsque le bloc fonction/DFB est appelé, les algorithmes définis par ce dernier ne sont pas exécutés et ENO est mis à « 0 ».

Au cas où la valeur d'EN est égale à « 1 », lorsque le bloc fonction/DFB est appelé, les algorithmes définis par ce dernier sont exécutés. Une fois les algorithmes exécutés, la valeur de la sortie ENO est réglée sur « 1 ». En cas d'erreur lors de l'exécution de ces algorithmes, la sortie ENO est réglée sur « 0 ».

Si aucune valeur n'est attribuée à la broche EN à l'appel du FFB, l'algorithme défini par ce dernier est exécuté (comme lorsque EN a la valeur « 1 »).

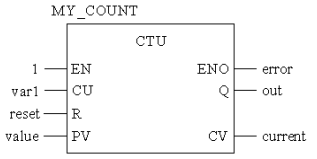
Si ENO est mis à « 0 » (du fait de EN = 0 ou d'une erreur d'exécution), les sorties du bloc fonction/DFB conservent l'état qu'elles avaient au dernier cycle exécuté correctement.

Le comportement aux sorties des blocs fonction/DFB est le même, que les blocs fonction/DFB aient été appelés sans EN/ENO ou avec EN = 1.

Si EN/ENO doivent être utilisés, l'appel du bloc fonction doit être exécuté sous forme d'appel formel. L'affectation d'une variable à ENO doit être effectuée avec l'opérateur =>.

```
CAL MY_COUNT (EN:=1, CU:=var1, R:=reset, PV:=value,
ENO=>error, Q=>out, CV=>current) ;
```

Appel du même bloc fonction dans FBD :



Variable VAR_IN_OUT

Très souvent, on utilise des blocs fonction pour lire une variable au niveau de l'entrée (variables d'entrée), traiter celle-ci, et sortir à nouveau les valeurs modifiées de la même variable (variables de sortie). Ce cas particulier d'une variable d'entrée/de sortie est également appelé variable VAR_IN_OUT.

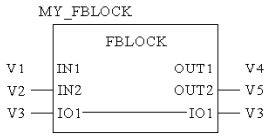
Il convient de noter les particularités suivantes lors de l'utilisation de blocs fonction/DFB avec des variables VAR_IN_OUT :

- une variable doit être affectée à toutes les entrées VAR_IN_OUT.
- il est interdit d'affecter des valeurs littérales ou des constantes aux entrées VAR_IN_OUT.
- **aucune** valeur ne doit être affectée aux sorties VAR_IN_OUT.
- les variables VAR_IN_OUT ne peuvent **pas** être utilisées en dehors de l'appel du bloc fonction.

Appel d'un bloc fonction avec une variable VAR_IN_OUT dans IL :

```
CAL MY_FBLOCK(IN1:=V1, IN2:=V2, IO1:=V3, OUT1=>V4, OUT2=>V5)
```

Appel du même bloc fonction dans FBD :



Les variables VAR_IN_OUT ne peuvent **pas** être utilisées en dehors de l'appel du bloc fonction.

Les appels de blocs fonction suivants sont par conséquent **non valides** :

Appel **non valide**, exemple 1 :

`LD V1`	Chargement de la variable `V1` dans l'accumulateur
`CAL InOutFB`	Appel d'un bloc fonction avec un paramètre `VAR_IN_OUT`. L'accumulateur est alors chargé avec référence à un paramètre `VAR_IN_OUT`.
`AND V2`	Liaison ET du contenu de l'accumulateur avec les variables `V2`. **Erreur :** l'opération ne peut pas être exécutée car il n'est pas possible d'accéder au paramètre `VAR_IN_OUT` (contenu de l'accumulateur) en dehors de l'appel d'un bloc fonction.

Appel **non valide**, exemple 2 :

`LD V1`	Chargement de la variable `V1` dans l'accumulateur
`AND InOutFB.inout`	Liaison ET du contenu de l'accumulateur avec référence à un paramètre `VAR_IN_OUT`. **Erreur :** l'opération ne peut pas être exécutée car il n'est pas possible d'accéder au paramètre `VAR_IN_OUT` en dehors de l'appel d'un bloc fonction.

Les appels de blocs fonction suivants sont en revanche **valides** :

Appel **valide**, exemple 1 :

`CAL InOutFB (IN1:=V1,inout:=V2`	Appel d'un bloc fonction avec un paramètre `VAR_IN_OUT` et affectation des paramètres réels au sein de l'appel de bloc fonction.

Appel **valide**, exemple 2 :

`LD V1`	Chargement de la variable `V1` dans l'accumulateur
`ST InOutFB.IN1`	Affectation du contenu de l'accumulateur au paramètre `IN1` du bloc fonction `IN1`.
`CAL InOutFB(inout:=V2)`	Appel du bloc fonction avec affectation du paramètre réel (`V2`) au paramètre `VAR_IN_OUT`.

Procédures d'appel

Procédure

Les procédures sont disponibles sous forme de bibliothèques. La logique des procédures est établie en langage de programmation C et ne peut pas être modifiée dans l'éditeur IL.

Comme les fonctions, les procédures n'ont pas d'états internes. Lorsque les valeurs d'entrée sont identiques, la valeur de sortie est la même à chaque exécution de la procédure. Par exemple, l'addition de deux valeurs donne toujours le même résultat.

Contrairement aux fonctions, les procédures ne livrent aucune valeur de renvoi et prennent en charge les variables `VAR_IN_OUT`.

Les procédures sont un complément de la norme CEI 61131-3 et doivent être activées de manière explicite.

Paramètres

Pour importer des valeurs dans une procédure ou exporter des valeurs d'une procédure, on a besoin d'« entrées » et de « sorties ». Ces entrées et ces sorties sont appelées « paramètres formels ».

Les états actuels du processus sont transmis aux paramètres formels. Ce sont les paramètres réels.

On peut utiliser comme paramètre réel des entrées de procédure :
- variable,
- adresse,
- valeur littérale.

On peut utiliser comme paramètre réel des sorties de procédure :
- variable,
- adresse.

Le type de données du paramètre réel doit correspondre au type de données du paramètre formel. La seule exception concerne les paramètres formels génériques dont le type de données est déterminé par le paramètre réel.

De plus, pour les paramètres formels génériques du type de données `ANY_BIT`, des paramètres réels du type de données `INT` ou `DINT` (pas `UINT` ni `UDINT`) peuvent être utilisés.

Il s'agit d'une extension de la norme CEI 61131-3, qui doit être activée de manière explicite.

Exemple :

Autorisé :

```
AND (AnyBitParam := IntVar1, AnyBitParam2 := IntVar2)
```

Non autorisé :

```
AND_WORD (WordParam1 := IntVar1, WordParam2 := IntVar2)
```

(Dans ce cas `AND_INT` doit être utilisé.)

```
AND_ARRAY_WORD (ArrayInt, ...)
```

(Dans ce cas, un changement de type explicite doit être effectué via `INT_ARR_TO_WORD_ARR (...)`.

Pour l'appel formel, il n'est en principe pas nécessaire d'affecter une valeur à tous les paramètres formels. Pour connaître les types de paramètre formel pour lesquels cela est cependant impératif, veuillez vous reporter au tableau.

Type de paramètre	EDT	STRING	ARRAY	ANY_ARRAY	IODDT	STRUCT	FB	ANY
Entrée	-	-	+	+	+	+	+	+
VAR_IN_OUT	+	+	+	+	+	+	/	+
Sortie	-	-	-	-	-	-	/	+
+ paramètre réel impératif								
- paramètre réel non impératif								
/ non applicable								

Si aucune valeur n'est affectée à un paramètre formel, la valeur initiale est utilisée pendant l'exécution du bloc fonction. Si aucune valeur initiale n'est définie, la valeur par défaut (0) est utilisée.

Remarques sur la programmation

Veuillez tenir compte des remarques qui suivent sur la programmation :

- Les procédures ne sont exécutées que lorsque l'entrée `EN` = 1 ou lorsque l'entrée `EN` est désactivée (voir aussi EN and ENO *(voir page 504)*).
- Des conditions particulières s'appliquent lors de l'utilisation de variables VAR_IN_OUT *(voir page 505)*.
- Il existe deux façons d'appeler une procédure :
  - appel formel (appel d'une fonction avec les noms des paramètres formels) Des variables peuvent alors être affectées aux sorties via l'opérateur `=>` (appel d'un bloc fonction sous forme abrégée).
  - appel informel (appel d'une fonction sans les noms des paramètres formels)

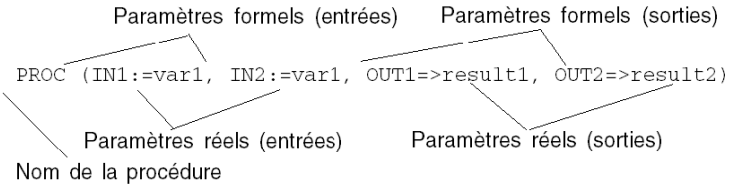
Appel formel

Avec cette forme d'appel (avec les noms des paramètres formels), les procédures sont appelées via une suite d'instructions composée d'une instruction optionnelle CAL suivie du nom de la procédure et d'une liste entre parenthèses des affectations de paramètres réels aux paramètres formels. L'affectation des paramètres formels des entrées s'effectue via l'affectation := et l'affectation des paramètres formels des sorties via l'affectation =>. L'ordre d'énumération des paramètres formels d'entrées et de sorties **n'est pas important**.

La liste des paramètres réels peut être éditée directement après une virgule.

Il est possible d'utiliser EN et ENO avec ce type d'appel.

Appel d'une procédure avec les noms des paramètres formels :



ou

```
CAL PROC (IN1:=var1, IN2:=var1, OUT1=>result1,OUT2=>result2)
```

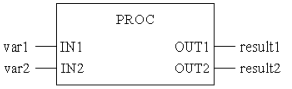
ou

```
PROC (IN1:=var1, IN2:=var1, OUT1=>result1, OUT2=>result2)
```

ou

```
CAL PROC (IN1:=var1, IN2:=var1, OUT1=>result1, OUT2=>result2)
```

Appel de la même procédure dans FBD :



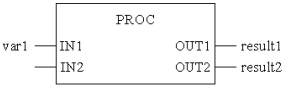
Lors d'un appel formel, il n'est pas nécessaire d'affecter une valeur à tous les paramètres formels (voir également Parameter *(voir page 500)*).

```
PROC (IN1:=var1, OUT1=>result1, OUT2=>result2)
```

ou

```
CAL PROC (IN1:=var1, OUT1=>result1, OUT2=>result2)
```

Appel de la même procédure dans FBD :

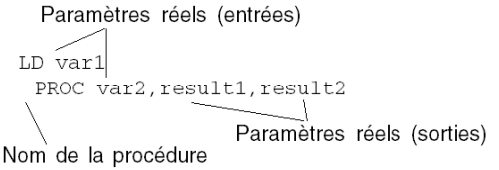


Appel informel sans instruction CAL

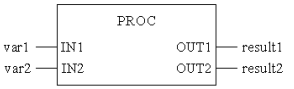
Avec cette forme d'appel (sans les noms des paramètres formels), les procédures sont appelées via une suite d'instructions qui comprend le chargement du premier paramètre réel dans l'accumulateur suivi du nom de la procédure, lui-même suivi d'une liste optionnelle des paramètres réels des entrées et sorties. L'ordre d'énumération des paramètres réels est **important**. La liste des paramètres réels ne peut pas être éditée.

EN et ENO ne peuvent **pas** être utilisés avec ce type d'appel.

Appel d'une procédure avec les noms des paramètres formels :



Appel de la même procédure dans FBD :



NOTE : veuillez noter que pour l'appel informel, la liste des paramètres réels ne doit **pas** être indiquée entre parenthèses. La norme CEI 61133-3 exige dans ce cas d'enlever les parenthèses, afin d'indiquer que le premier paramètre réel ne fait pas partie de la liste.

Appel informel **non valide** d'une procédure :

```
LD A
LIMIT (B,C)
```

Si la valeur à traiter (premier paramètre réel) se trouve déjà dans l'accumulateur, l'instruction de chargement n'est plus nécessaire.

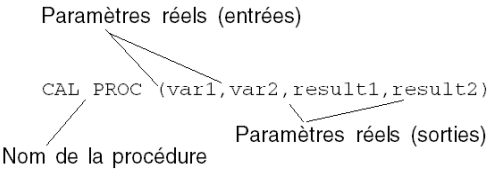
```
EXAMP1 var2,result1,result2
```

Appel informel avec instruction CAL

Avec cette forme d'appel, les procédures sont appelées par une suite d'instructions composée de l'instruction CAL suivie du nom de la procédure, suivi lui-même de la liste entre parenthèses des paramètres réels des entrées et sorties. L'ordre d'énumération des paramètres réels est **important**. La liste des paramètres réels ne peut pas être éditée.

EN et ENO ne peuvent **pas** être utilisés avec ce type d'appel.

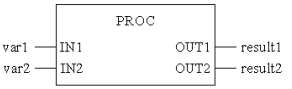
Appel d'une procédure avec les noms de paramètres formels et l'instruction CAL :



ou

```
CAL PROC (var1, var2, result1, result2)
```

Appel de la même procédure dans FBD :



NOTE : contrairement à l'appel informel sans instruction CAL, dans le cadre de l'appel informel avec instruction CAL, la valeur à traiter (le premier paramètre réel) n'est pas chargée explicitement dans l'accumulateur, mais fait partie de la liste des paramètres réels. Par conséquent, lors d'appels informels à l'aide d'une instruction CAL, la liste des paramètres réels doit être indiquée entre parenthèses.

EN et ENO

Pour toutes les procédures, une entrée EN et une sortie ENO peuvent être configurées.

Si la valeur d'EN est égale à « 0 », lorsque la procédure est appelée, les algorithmes définis par cette dernière ne sont pas exécutés et ENO est mis sur « 0 ».

Si la valeur d'EN est égale à « 1 », lorsque la procédure est appelée, les algorithmes définis par la procédure sont exécutés. Une fois les algorithmes exécutés, la valeur de la sortie ENO est réglée sur « 1 ». En cas d'erreur lors de l'exécution de ces algorithmes, la sortie ENO est réglée sur « 0 ».

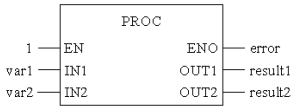
Si aucune valeur n'est attribuée à la broche EN à l'appel du FFB, l'algorithme défini par ce dernier est exécuté (comme lorsque EN a la valeur « 1 »).

Si ENO est mis sur « 0 » (en raison de EN=0 ou d'une erreur d'exécution), les sorties de la procédure sont mises sur « 0 ».

Si EN/ENO doivent être utilisés, l'appel de la procédure doit être exécuté comme un appel formel. L'affectation d'une variable à ENO doit être effectuée avec l'opérateur =>.

```
PROC (EN:=1, IN1:=var1, IN2:=var2, ENO=>error,
OUT1=>result1, OUT2=>result2) ;
```

Appel de la même procédure dans FBD :



Variable VAR_IN_OUT

Très souvent, on utilise des procédures pour lire une variable au niveau de l'entrée (variables d'entrée), traiter celle-ci, et sortir à nouveau les valeurs modifiées de la même variable (variables de sortie). Ce cas particulier d'une variable d'entrée/de sortie est également appelé variable `VAR_IN_OUT`.

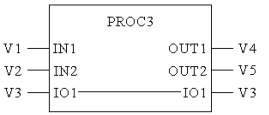
Il convient de noter les particularités suivantes dans le cas de l'utilisation de procédures avec des variables `VAR_IN_OUT` :

- une variable doit être affectée à toutes les entrées `VAR_IN_OUT`.
- il est interdit d'affecter des valeurs littérales ou des constantes aux entrées `VAR_IN_OUT`.
- **aucune** valeur ne doit être affectée aux sorties `VAR_IN_OUT`.
- les variables `VAR_IN_OUT` ne peuvent **pas** être utilisées en dehors de l'appel de procédure.

Appel d'une procédure avec une variable `VAR_IN_OUT` dans IL :

```
PROC3 (IN1:=V1, IN2:=V2, IO1:=V3, OUT1=>V4, OUT2=>V5) ;
```

Appel de la même procédure dans FBD :



Les variables `VAR_IN_OUT` ne peuvent **pas** être utilisées en dehors de l'appel de procédure.

Les appels de procédure suivants sont par conséquent **invalides** :

Appel **non valide**, exemple 1 :

`LD V1`	Chargement de la variable `V1` dans l'accumulateur
`CAL InOutProc`	Appel d'une procédure avec un paramètre `VAR_IN_OUT`. L'accumulateur est alors chargé avec référence à un paramètre `VAR_IN_OUT`.
`AND V2`	Liaison ET du contenu de l'accumulateur avec la variable `V2`. **Erreur :** l'opération ne peut pas être exécutée car il n'est pas possible d'accéder au paramètre `VAR_IN_OUT` (contenu de l'accumulateur) en dehors de l'appel de procédure.

Appel **non valide**, exemple 2 :

`LD V1`	Chargement de la variable `V1` dans l'accumulateur
`AND InOutProc.inout`	Liaison ET du contenu de l'accumulateur avec référence à un paramètre `VAR_IN_OUT`. **Erreur :** l'opération ne peut pas être exécutée car il n'est pas possible d'accéder au paramètre `VAR_IN_OUT` en dehors de l'appel de procédure.

Appel **non valide**, exemple 3 :

`LD V1`	Chargement de la variable `V1` dans l'accumulateur
`InOutFB V2`	Appel de la procédure avec affectation du paramètre réel (`V2`) au paramètre `VAR_IN_OUT`. **Erreur :** l'opération ne peut pas être exécutée car, pour ce type d'appel de procédure, le paramètre `VAR_IN_OUT` continuerait d'être utilisable dans l'accumulateur.

Les appels de procédure suivants sont en revanche **valides** :

Appel **valide**, exemple 1 :

`CAL InOutProc (IN1:=V1,inout:=V2)`	Appel d'une procédure avec un paramètre `VAR_IN_OUT` et affectation formelle des paramètres réels au sein de l'appel de procédure.

Appel **valide**, exemple 2 :

`InOutProc (IN1:=V1,inout:=V2)`	Appel d'une procédure avec un paramètre `VAR_IN_OUT` et affectation formelle des paramètres réels au sein de l'appel de procédure.

Appel **valide**, exemple 3 :

`CAL InOutProc (V1,V2)`	Appel d'une procédure avec un paramètre `VAR_IN_OUT` et affectation informelle des paramètres réels au sein de l'appel de procédure.

Texte structuré (ST)

15

Objet de ce sous-chapitre

Ce chapitre décrit le langage de programmation Littéral structuré ST conforme à la norme CEI 61131.

Contenu de ce chapitre

Ce chapitre contient les sous-chapitres suivants :

15.1 Remarques générales sur le littéral structuré ST

Objet de ce sous-chapitre

Ce sous-chapitre vous donne un aperçu général sur le littéral structuré ST.

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Informations générales sur le texte structuré (ST)

Présentation

Le langage Littéral structuré (ST) vous permet par exemple d'appeler des blocs fonction, d'exécuter des fonctions, de lancer des affectations, d'exécuter des instructions conditionnelles et de répéter des instructions.

Expression

Le langage ST utilise ce que l'on appelle des "expressions".

Les expressions sont des constructions comprenant opérateurs et opérandes qui livrent une valeur lors de leur exécution.

Opérateur

Les opérateurs sont des symboles pour les opérations à exécuter.

Opérande

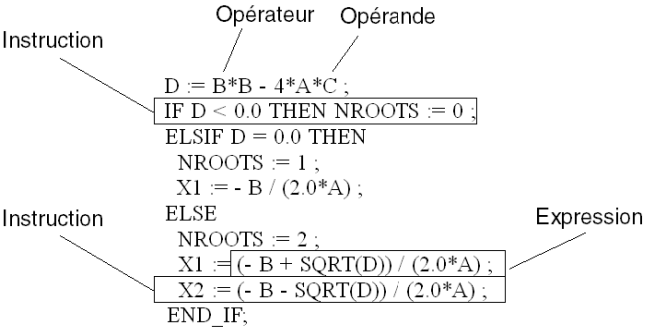
Les opérateurs sont utilisés sur les opérandes. Les opérandes sont par exemple des variables, des valeurs littérales, des entrées/sorties FFB, etc.

Instructions

Les instructions permettent d'affecter les valeurs retournées par des expressions à des paramètres réels et de structurer et commander les expressions.

Représentation d'une section ST

Représentation d'une section ST :



Taille de la section

La taille d'une ligne d'instruction est limitée à 300 caractères.

La longueur d'une section ST n'est pas limitée au sein de l'environnement de programmation. La longueur d'une section ST n'est limitée que par la taille de la mémoire de l'automate.

Syntaxe

Il n'est pas fait de différence entre majuscules et minuscules lors de la saisie des identificateurs et des mots-clés.

Exception : Les caractères d'espacement et de tabulation ne sont pas autorisés dans :

- les mots-clés,
- les valeurs littérales,
- les valeurs,
- les identificateurs,
- les variables et
- combinaisons du limiteur [par ex., `(*` pour commentaires)].

Ordre d'exécution

L'interprétation d'une expression consiste à appliquer les opérateurs aux opérandes, selon la séquence qui est définie par le rang des opérateurs (voir Tableau des opérateurs *(voir page 514)*). Le système exécute d'abord l'opérateur présentant le rang le plus élevé dans l'expression, suivi de l'opérateur du rang inférieur suivant, etc. jusqu'à ce que l'opération soit terminée. Les opérateurs de même rang sont exécutés de gauche à droite, comme ils sont écrits dans l'expression. Cet ordre de traitement peut être modifié en utilisant des parenthèses.

Si par exemple `A`, `B`, `C` et `D` ont respectivement les valeurs 1, 2, 3 et 4, et que le calcul est effectué comme suit :

```
A+B-C*D
```

alors le résultat sera -9.

Pour un calcul tel que :

```
(A+B-C)*D
```

le résultat sera 0.

Si un opérateur a deux opérandes, l'opérande gauche est exécuté en premier. Par exemple, dans l'expression

```
SIN(A)*COS(B)
```

l'expression `SIN(A)` est calculée en premier, puis c'est au tour de `COS(B)` et enfin le produit est calculé.

Comportement en cas d'erreur

Les conditions suivantes seront traitées comme des erreurs lors de l'exécution d'une expression, par exemple :

- tentative de division par 0.
- les opérandes n'ont pas le type de données correct pour l'opération.
- le résultat d'une opération numérique dépasse la plage de valeurs de son type de données.

Si une erreur se produit lors de l'exécution d'une opération, le bit système correspondant (%S) est activé (si cela est pris en charge par l'automate utilisé).

Conformité CEI

Pour plus d'informations sur la conformité CEI du langage ST, voir Conformité CEI *(voir page 657)*.

Opérandes

Présentation

Un opérande peut être :
- une adresse,
- un libellé,
- une variable,
- une variable multi-éléments,
- un élément d'une variable multi-éléments,
- un appel de fonction ou
- une sortie FFB.

Types de données

Les types de données des opérandes à traiter dans une instruction doivent être identiques. Si des opérandes de différents types de données doivent être traités, une conversion de types doit obligatoirement être effectuée auparavant.

Dans l'exemple, la variable Integer `i1` est convertie en une variable Real, avant d'être ajoutée à la variable Real `r4`.

```
r3 := r4 + SIN(INT_TO_REAL(i1)) ;
```

Comme exception à cette règle, des variables du type de données `TIME` peuvent être multipliées par des variables du type de données `INT`, `DINT`, `UINT` ou `UDINT` ou divisées par ces dernières.

Opérations autorisées :
- `timeVar1 := timeVar2 / dintVar1;`
- `timeVar1 := timeVar2 * intVar1;`
- `timeVar := 10 * time#10s;`

Cette fonction est considérée comme " indésirable " par la norme CEI 61131-3.

Utilisation directe d'adresses

Les adresses peuvent être utilisées directement (sans déclaration préalable). Dans ce cas, le type de données est directement affecté à l'adresse. L'affectation a lieu via le "préfixe de taille".

Le tableau suivant indique les différents préfixes de taille :

Préfixe de taille / Symbole	Exemple	Type de données
pas de préfixe	%I10, %CH203.MOD, %CH203.MOD.ERR	BOOL
X	%M**X**20	BOOL
B	%Q**B**102.3	BYTE
W	%K**W**43	INT
D	%Q**D**100	DINT
F	%M**F**100	REAL

Utilisation d'autres types de données

Si d'autres types de données doivent être affectés en tant que types de données par défaut d'une adresse, cela doit faire l'objet d'une déclaration explicite. L'éditeur de variables facilite la déclaration de ces variables. Il n'est pas possible de déclarer directement le type de données d'une adresse dans une section ST (par ex. la déclaration `AT %MW1 : UINT` ; non permise).

Exemple : les variables ci-dessous sont déclarées dans l'éditeur de variables.

```
UnlocV1 : ARRAY [1..10] OF INT;
LocV1 :   ARRAY [1..10] OF INT AT %MW100;
LocV2 :   TIME AT %MW100;
```

Les appels ci-dessous sont donc corrects du point de vue de la syntaxe :

```
%MW200 := 5;
UnlocV1[2] := LocV1[%MW200];
LocV2      := t#3s;
```

Accès aux variables de champs

Lors d'un accès aux variables de champ (`ARRAY`), seuls les libellés et les variables du type `INT`, `UINT`, `DINT` et `UDINT` sont autorisés dans l'indication d'index.

L'index d'un élément ARRAY peut être négatif si la limite inférieure de la plage est négative.

Exemple : Emploi de variables de zone

```
var1[i] := 8 ;
var2.otto[4] := var3 ;
var4[1+i+j*5] := 4 ;
```

Opérateurs

Présentation

Un opérateur est un symbole pour :
- une opération arithmétique à effectuer ou
- une opération logique à exécuter ou
- un traitement de fonction (appel)

Les opérateurs sont génériques, ce qui signifie qu'ils s'adaptent automatiquement au type de données de l'opérande.

Tableau des opérateurs

Les opérateurs sont exécutés en fonction de leur rang, voir également *Ordre d'exécution, page 510*.

Opérateurs du langage ST :

Opérateur	Signification	Rang	Opérandes possibles	Description

`()`	Mise entre parenthèses	1 (le plus haut)	Expression	La mise entre parenthèses est utilisée pour modifier la séquence d'exécution des opérateurs.
				Exemple : Si les opérandes `A`, `B`, `C` et `D` ont respectivement les valeurs 1, 2, 3 et 4, alors
				`A+B-C*D`
				donne le résultat -9 et
				`(A+B-C)*D`
				donne le résultat 0.

`FUNCNAME` (liste des paramètres réels)	Traitement de fonction (appel)	2	Expression, valeur littérale, variable, adresse (tous les types de données)	Le traitement de fonction est utilisé pour exécuter des fonctions (voir *Appel de fonctions élémentaires, page 542*).
-	Négation	3	Expression, valeur littérale, variable, adresse du type de données `INT`, `DINT`, `UINT`, `UDINT` ou `REAL`	Dans le cas de la négation `-`, le système change le signe de la valeur de l'opérande. **Exemple :** Dans l'exemple, `OUT` a la valeur -4 si `IN1` est égal à 4. `OUT := - IN1 ;`
`NON`	Complément	3	Expression, valeur littérale, variable, adresse du type de données `BOOL`, `BYTE`, `WORD` ou `DWORD`	Dans le cas de `NOT`, le système effectue une inversion de chaque bit de l'opérande. **Exemple :** Dans l'exemple, `OUT` a la valeur 0011001100 si `IN1` est égal à 1100110011. `OUT := NOT IN1 ;`

Opérateur	Signification	Rang	Opérandes possibles	Description
**	Elévation de puissance	4	Expression, valeur littérale, variable, adresse du type de données REAL (base) et INT, DINT, UINT, UDINT ou REAL (exposant)	Dans le cas de l'élévation à une puissance plus haute **, la valeur du premier opérande (base) est augmentée de la valeur du second opérande (exposant). **Exemple :** Dans l'exemple, OUT est égal à 625,0 si IN1 est 5,0 et IN2 4,0. `OUT := IN1 ** IN2 ;`
*	Multiplication	5	Expression, valeur littérale, variable, adresse du type de données INT, DINT, UINT, UDINT ou REAL	Dans le cas de la multiplication *, la valeur du premier opérande est multipliée par la valeur du deuxième opérande. **Exemple :** Dans l'exemple, OUT est égal à 20,0 si IN1 est 5,0 et IN2 4,0. `OUT := IN1 * IN2 ;` **Remarque :** La fonction MULTIME de la bibliothèque obsolète est destinée aux multiplications du type de données Time.
/	Division	5	Expression, valeur littérale, variable, adresse du type de données INT, DINT, UINT, UDINT ou REAL	Dans le cas de la division /, la valeur du premier opérande est divisée par la valeur du deuxième opérande. **Exemple :** Dans l'exemple, OUT est égal à 4,0 si IN1 est 20,0 et IN2 5,0. `OUT := IN1 / IN2 ;` **Remarque :** La fonction DIVTIME de la bibliothèque obsolète est destinée aux divisions du type de données Time.
MOD	Modulo	5	Expression, valeur littérale, variable, adresse du type de données INT, DINT, UINT ou UDINT	Dans le cas de MOD, la valeur du premier opérande est divisée par la valeur du deuxième opérande et le reste de la division (Modulo) est sorti comme résultat. **Exemple :** Dans l'exemple donné, • OUT est 1 si IN1 est 7 et IN2 2 • OUT est 1, si IN1 est 7 et IN2 -2 • OUT est -1, si IN1 est -7 et IN2 2 • OUT est -1, si IN1 est -7 et IN2 -2 `OUT := IN1 MOD IN2 ;`
+	Addition	6	Expression, valeur littérale, variable, adresse du type de données INT, DINT, UINT, UDINT, REAL ou TIME	Dans le cas de l'addition +, la valeur du premier opérande est ajoutée à la valeur du deuxième opérande. **Exemple :** Dans l'exemple donné, OUT est égal à 9 si IN1 est 7 et IN2 2. `OUT := IN1 + IN2 ;`

Opérateur	Signification	Rang	Opérandes possibles	Description
-	Soustraction	6	Expression, valeur littérale, variable, adresse du type de données INT, DINT, UINT, UDINT, REAL ou TIME	Dans le cas de la soustraction -, la valeur du deuxième opérande est soustraite à la valeur du premier opérande. **Exemple :** Dans l'exemple, OUT est égal à 6 si IN1 est 10 et IN2 4. `OUT := IN1 - IN2 ;`
<	Comparaison "inférieur à"	7	Expression, valeur littérale, variable, adresse du type de données BOOL, BYTE, INT, DINT, UINT, UDINT, REAL, TIME, WORD, DWORD, STRING, DT, DATE ou TOD	< permet de comparer la valeur du premier opérande à celle du deuxième opérande. Si la valeur du premier opérande est inférieure à celle du second, le résultat est un 1 booléen. Si la valeur du premier opérande est supérieure ou égale à celle du second, le résultat est un 0 booléen. **Exemple :** Dans l'exemple, OUT est égal à 1 si IN1 est inférieur à 10. Sinon, il est sur 0. `OUT := IN1 < 10 ;`
>	Comparaison "supérieur à"	7	Expression, valeur littérale, variable, adresse du type de données BOOL, BYTE, INT, DINT, UINT, UDINT, REAL, TIME, WORD, DWORD, STRING, DT, DATE ou TOD	> permet de comparer la valeur du premier opérande à celle du deuxième opérande. Si la valeur du premier opérande est supérieure à celle du second, le résultat est un 1 booléen. Si la valeur du premier opérande est inférieure ou égale à celle du second, le résultat est un 0 booléen. **Exemple :** Dans l'exemple, OUT est égal à 1 si IN1 est supérieur à 10. Si IN1 est inférieur à 0, alors il est sur 0. `OUT := IN1 > 10 ;`
<=	Comparaison "inférieur ou égal à"	7	Expression, valeur littérale, variable, adresse du type de données BOOL, BYTE, INT, DINT, UINT, UDINT, REAL, TIME, WORD, DWORD, STRING, DT, DATE ou TOD	<= permet de comparer la valeur du premier opérande à celle du deuxième opérande. Si la valeur du premier opérande est inférieure ou égale à celle du second, le résultat est un 1 booléen. Si la valeur du premier opérande est supérieure à celle du second, le résultat est un 0 booléen. **Exemple :** Dans l'exemple, OUT est égal à 1 si IN1 est inférieur ou égal à 10. Sinon, il est égal à 0. `OUT := IN1 <= 10 ;`

Opérateur	Signification	Rang	Opérandes possibles	Description
>=	Comparaison "supérieur ou égal à"	7	Expression, valeur littérale, variable, adresse du type de données BOOL, BYTE, INT, DINT, UINT, UDINT, REAL, TIME, WORD, DWORD, STRING, DT, DATE ou TOD	>= permet de comparer la valeur du premier opérande à celle du deuxième opérande. Si la valeur du premier opérande est supérieure ou égale à celle du second, le résultat est un 1 booléen. Si la valeur du premier opérande est inférieure à celle du second, le résultat est un 0 booléen. **Exemple :** Dans l'exemple, OUT est égal à 1 si IN1 est supérieur ou égal à 10. Sinon, il est égal à 0. `OUT := IN1 >= 10 ;`
=	Egalité	8	Expression, valeur littérale, variable, adresse du type de données BOOL, BYTE, INT, DINT, UINT, UDINT, REAL, TIME, WORD, DWORD, STRING, DT, DATE ou TOD	= permet de comparer la valeur du premier opérande à celle du deuxième opérande. Si la valeur du premier opérande est égale à celle du second, le résultat est un 1 booléen. Si la valeur du premier opérande est différente de celle du second, le résultat est un 0 booléen. **Exemple :** Dans l'exemple, OUT est égal à 1 si IN1 est égal à 10. Sinon, il est égal à 0. `OUT := IN1 = 10 ;`
<>	Inégalité	8	Expression, valeur littérale, variable, adresse du type de données BOOL, BYTE, INT, DINT, UINT, UDINT, REAL, TIME, WORD, DWORD, STRING, DT, DATE ou TOD	<> permet de comparer la valeur du premier opérande à celle du deuxième opérande. Si la valeur du premier opérande est différente de celle du second, le résultat est un 1 booléen. Si la valeur du premier opérande est égale à celle du second, le résultat est un 0 booléen. **Exemple :** Dans l'exemple, OUT est égal à 1 si IN1 n'est pas égal à 10. Sinon, il est sur 0. `OUT := IN1 <> 10 ;`
&	ET logique	9	Expression, valeur littérale, variable, adresse du type de données BOOL, BYTE, WORD ou DWORD	& permet d'établir une liaison ET logique entre les opérandes. Pour les types de données BYTE, WORD et DWORD, le lien est fait par bit. **Exemple :** Dans les exemples, OUT est égal à 1 si IN1, IN2 et IN3 sont sur 1. `OUT := IN1 & IN2 & IN3 ;`
AND	ET logique	9	Expression, valeur littérale, variable, adresse du type de données BOOL, BYTE, WORD ou DWORD	AND permet d'établir une liaison ET logique entre les opérandes. Pour les types de données BYTE, WORD et DWORD, le lien est fait par bit. **Exemple :** Dans les exemples, OUT est égal à 1 si IN1, IN2 et IN3 sont sur 1. `OUT := IN1 AND IN2 AND IN3 ;`

Opérateur	Signification	Rang	Opérandes possibles	Description

XOR	OU exclusif logique	10	Expression, valeur littérale, variable, adresse du type de données BOOL, BYTE, WORD ou DWORD	XOR permet d'établir une liaison OU exclusif logique entre les opérandes. Pour les types de données BYTE, WORD et DWORD, le lien est fait par bit.
				Exemple : Dans l'exemple, OUT est sur 1 si IN1 et IN2 ne sont pas égaux. Si IN1 et IN2 ont le même état (tous deux 0 ou 1), OUT est sur 0.
				`OUT := IN1 XOR IN2 ;`
				Si plus de deux opérandes sont reliés, le résultat de l'opération est à l'état 1 pour un nombre impair d'états 1 et à l'état 0 pour un nombre pair d'états 1.
				Exemple : Dans l'exemple, OUT est égal à 1 si 1 ou 3 opérandes sont sur 1. OUT est sur 0 si 0, 2 ou 4 opérandes sont sur 1.
				`OUT := IN1 XOR IN2 XOR IN3 XOR IN4 ;`

OR	OU logique	11 (le plus bas)	Expression, valeur littérale, variable, adresse du type de données BOOL, BYTE, WORD ou DWORD	OR permet d'établir une liaison OU logique entre les opérandes. Pour les types de données BYTE, WORD et DWORD, le lien est fait par bit. **Exemple :** Dans l'exemple, OUT est sur 1 si IN1, IN2 ou IN3 est sur 1. `OUT := IN1 OR IN2 OR IN3 ;`

15.2 Instructions

Objet de ce sous-chapitre

Ce sous-chapitre décrit les instructions du langage de programmation Littéral structuré ST.

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Instructions

Description

Les instructions sont les "commandes" du langage de programmation ST.

Les instructions doivent être terminées par des points-virgules.

Une ligne peut contenir plusieurs instructions (séparées par des points-virgules).

Un seul point-virgule représente une instruction d'espacement *(voir page 538)*.

Affectation

Présentation

L'affectation remplace la valeur courante d'une variable à élément unique ou multiple par le résultat de l'évaluation d'une expression.

Une affectation est composée d'une indication de variables à gauche, suivie de l'opérateur d'affectation :=, suivi de l'expression à évaluer.

Les deux variables (côtés gauche et droit de l'opérateur d'affectation) doivent être du même type de données.

Les variables ARRAY font exception. A l'issue de l'activation explicite de l'option correspondante, l'affectation de deux variables ARRAY ayant des longueurs différentes est possible.

Affectation de la valeur d'une variable à une autre variable

Les affectations sont utilisées pour affecter la valeur d'une variable à une autre variable.

L'instruction

```
A := B ;
```

est par exemple utilisée pour remplacer la valeur de la variable A par la valeur courante de la variable B. Si A et B ont un type de données élémentaire, la valeur individuelle de B est transmise vers A. Si A et B ont un type de données dérivé, les valeurs de tous les éléments de B sont transmises vers A.

Affectation d'une valeur littérale à une variable

Les affectations sont utilisées pour affecter une valeur littérale à une variable.

L'instruction

```
C := 25 ;
```

est par exemple utilisée pour affecter la valeur 25 à la variable C.

Affectation de la valeur d'une opération à une variable

Les affectations sont utilisées pour affecter à une variable une valeur qui est le résultat d'une opération.

L'instruction

```
X := (A+B-C)*D ;
```

est par exemple utilisée pour affecter à la variable X le résultat de l'opération (A+B-C)*D.

Affectation de la valeur d'un FFB à une variable

Les affectations sont utilisées pour affecter à une variable une valeur renvoyée par une fonction ou un bloc fonction.

L'instruction

```
B := MOD(C,A) ;
```

est par exemple utilisée pour appeler la fonction `MOD` (modulo) et affecter le résultat du calcul à la variable `B`.

L'instruction

```
A := MY_TON.Q ;
```

est par exemple utilisée pour affecter à la variable `A` la valeur de la sortie `Q` du bloc fonction `MY_TON` (instance du bloc fonction `TON`). (Il ne s'agit pas d'un appel de bloc fonction))

Affectations multiples

Les affectations multiples sont une extension de la norme CEI 61131-3 et doivent être activées de manière explicite.

Même à l'issue de l'activation, les affectations multiples ne sont PAS autorisées dans les cas suivants :

- dans la liste de paramètres d'un appel de bloc fonction
- dans la liste d'éléments pour l'initialisation de variables structurées

L'instruction

```
X := Y := Z
```

est permise.

Les instructions

```
FB(in1 := 1, In2 := In3 := 2) ;
```

et

```
strucVar := (comp1 := 1, comp2 := comp3 := 2) ;
```

ne sont pas permises.

Affectations entre variables ARRAY et WORD/DWORD.

Les affectations entre variables ARRAY et WORD/DWORD ne sont possibles que si une conversion de types a été effectuée au préalable, par ex. :

```
%Q3.0:16 := INT_TO_AR_BOOL(%MW20) ;
```

Les fonctions de conversion suivantes sont disponibles (bibliothèque générale, famille ARRAY) :

- MOVE_BOOL_AREBOOL
- MOVE_WORD_ARWORD
- MOVE_DWORD_ARDWORD
- MOVE_INT_ARINT
- MOVE_DINT_ARDINT
- MOVE_REAL_ARREAL

Sélectionner l'instruction IF...THEN...END_IF

Description

L'instruction IF signifie qu'une instruction ou un groupe d'instructions peuvent être seulement exécutés si l'expression booléenne correspondante a la valeur 1 (vrai). Si la condition a pour valeur 0 (faux), l'instruction ou le groupe d'instructions ne sont pas exécutés.

L'instruction `THEN` marque la fin d'une condition et le début d'une instruction (des instructions).

L'instruction `END_IF` marque la fin de l'instruction (des instructions).

NOTE : Vous pouvez imbriquer autant d'instructions `IF...THEN...END_IF` que vous voulez pour créer des instructions de sélection complexes.

Exemple `IF...THEN...END_IF`

La condition peut être exprimée via une variable booléenne.

Si `FLAG` a la valeur 1, les instructions sont exécutées, si `FLAG` a la valeur 0, elles ne sont pas exécutées.

```
IF FLAG THEN
   C:=SIN(A) * COS(B) ;
   B:=C - A ;
END_IF ;
```

La condition peut également être exprimée via une opération qui livre un résultat booléen.

Si `A` est supérieur à `B`, les instructions sont exécutées ; si `A` est inférieur ou égal à `B`, elles ne sont pas exécutées.

```
IF A>B THEN
   C:=SIN(A) * COS(B) ;
   B:=C - A ;
END_IF ;
```

Exemple `IF NOT...THEN...END_IF`

`NOT` permet d'inverser la condition (les deux instructions sont exécutées si le résultat est 0).

```
IF NOT FLAG THEN
   C:=SIN_REAL(A) * COS_REAL(B) ;
   B:=C - A ;
END_IF ;
```

Voir également

ELSE *(voir page 526)*

ELSIF *(voir page 527)*

Sélectionner l'instruction ELSE

Description

L'instruction ELSE vient toujours après une instruction IF...THEN, ELSIF...THEN ou CASE.

Si l'instruction ELSE vient après IF ou ELSIF, l'instruction ou le groupe d'instructions sont exécutés seulement si les expressions booléennes correspondantes des instructions IF et ELSIF ont la valeur 0 (faux). Si la condition de l'instruction IF ou ELSIF est 1 (vrai), l'instruction ou le groupe d'instructions ne sont pas exécutés.

Si l'instruction ELSE vient après CASE, l'instruction ou le groupe d'instructions ne sont exécutés que si aucun repère ne contient la valeur du sélecteur. Si un repère contient la valeur du sélecteur, l'instruction ou le groupe d'instructions ne sont pas exécutés.

NOTE : Vous pouvez imbriquer autant d'instructions IF...THEN...ELSE...END_IF que vous voulez pour créer des instructions de sélection complexes.

Exemple ELSE

```
IF A>B THEN
   C:=SIN(A) * COS(B) ;
   B:=C - A ;
ELSE
   C:=A + B ;
   B:=C * A ;
END_IF ;
```

Voir également

IF *(voir page 524)*

ELSIF *(voir page 527)*

CASE *(voir page 529)*

Instruction de sélection ELSIF...THEN

Description

L'instruction ELSIF vient toujours après une instruction IF...THEN. L'instruction ELSIF détermine qu'une instruction ou un groupe d'instructions sont exécutés seulement si l'expression booléenne correspondante de l'instruction IF a la valeur 0 (faux) et que l'expression booléenne correspondante de l'instruction ELSIF a la valeur 1 (vrai). Si la condition de l'instruction IF est 1 (vrai) ou que la condition de l'instruction ELSIF est 0 (faux), l'instruction ou le groupe d'instructions ne sont pas exécutés.

L'instruction THEN caractérise la fin de la (des) condition(s) ELSIF et le début de l'instruction (des instructions).

NOTE : Vous pouvez imbriquer autant d'instructions IF...THEN...ELSIF...THEN...END_IF que vous voulez pour créer des instructions de sélection complexes.

Exemple ELSIF...THEN

```
IF A>B THEN
   C:=SIN(A) * COS(B) ;
   B:=SUB(C,A) ;
ELSIF A=B THEN
   C:=ADD(A,B) ;
   B:=MUL(C,A) ;
END_IF ;
```

Exemple d'instructions imbriquées

```
IF A>B THEN
     IF B=C THEN
         C:=SIN(A) * COS(B) ;
     ELSE
         B:=SUB(C,A) ;
     END_IF ;
ELSIF A=B THEN
     C:=ADD(A,B) ;
     B:=MUL(C,A) ;
ELSE
     C:=DIV(A,B) ;
END_IF ;
```

Voir également

IF *(voir page 524)*

ELSE *(voir page 526)*

Sélection de l'instruction CASE...OF...END_CASE

Description

L'instruction CASE est composée d'une expression de type données INT (le "sélecteur") et d'une liste de groupes d'instructions. Chaque groupe porte un repère composé d'un ou de plusieurs entiers (INT, DINT, UINT, UDINT) ou de plages de valeurs entières. L'on exécutera le premier groupe d'instructions dont le repère contient la valeur calculée du sélecteur. Sinon aucune des instructions n'est exécutée.

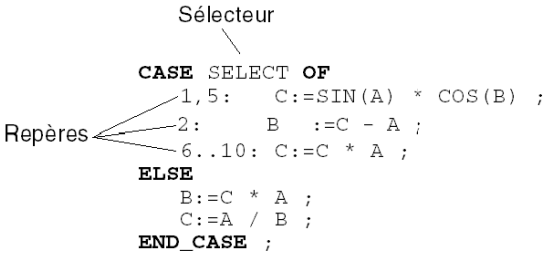
L'instruction OF caractérise le début des repères.

A l'intérieur d'une instruction CASE, on peut définir une instruction ELSE dont les instructions seront exécutées si aucun repère ne contient la valeur du sélecteur.

L'instruction END_CASE marque la fin de l'instruction (des instructions).

Exemple CASE...OF...END_CASE

Exemple CASE...OF...END_CASE



Voir également

ELSE *(voir page 526)*

Instruction récurrente FOR...TO...BY...DO...END_FOR

Description

L'instruction FOR est utilisée si le nombre d'occurrences peut être défini à l'avance. Sinon, on utilise WHILE *(voir page 533)* ou REPEAT *(voir page 534)*.

L'instruction FOR reprend une chaîne d'instructions jusqu'à l'instruction END_FOR. Le nombre d'occurrences est déterminé par la valeur initiale, la valeur finale et la variable de commande.

Variable de commande, valeur initiale et valeur finale doivent avoir le même type de données (DINT ou INT).

Variable de commande, variable initiale et variable finale peuvent être modifiées via une des instructions récurrentes. Cela constitue un complément de la norme CEI 61131-3.

L'instruction FOR incrémente la valeur des variables de commande d'une valeur initiale à une valeur finale. Par défaut, la valeur de l'incrément est définie sur 1. Pour le cas où une autre valeur doit être utilisée, il est possible d'indiquer explicitement une valeur d'incrément (variable ou constante). La valeur des variables de commande est contrôlée avant chaque nouveau parcours de la boucle. La boucle est abandonnée si ladite valeur est en dehors de la plage de la valeur initiale et de la valeur finale.

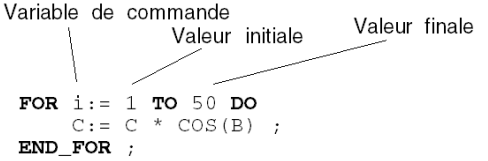
Avant le premier parcours de la boucle, on contrôlera si l'incrément des variables de commande progresse vers la valeur finale à partir de la valeur initiale. Si ce n'est pas le cas (par exemple valeur initiale ≤valeur finale et incrément négatif), la boucle n'est pas traitée. La valeur de la variable de commande n'est pas définie en dehors de la boucle.

L'instruction DO marque la fin de définition d'une occurrence et le début d'une instruction (des instructions).

La répétition peut être prématurément quittée avec l'instruction EXIT. L'instruction END_FOR marque la fin de l'instruction (des instructions).

Exemple : FOR avec incrément 1

FOR avec incrément 1

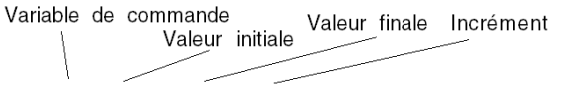


FOR avec incrément différent de 1

Si un autre incrément que 1 doit être utilisé vous pouvez le définir avec BY. Incrément, valeur initiale, valeur finale et variable de commande doivent avoir le même type de données (DINT ou INT). Le critère qui détermine le sens de traitement (comptage, décomptage) est le signe de l'expression BY. Si cette expression est positive, la boucle est comptée, si elle est négative, la boucle est décomptée.

Exemple : Comptage à deux étapes

comptage à deux étapes




```
FOR i:= 1 TO 10 BY 2 DO (* BY > 0 : Boucle   vers   l'avant *)
  C:= C * COS(B) ; (* l'instruction est exécutée 5 x *)
END_FOR ;
```

Exemple : Décomptage

décomptage

```
FOR i:= 10 TO 1 BY -1 DO (* BY < 0 : boucle décomptée *)
  C:= C * COS(B) ; (* Instr. est exécutée 10 x *)
END_FOR ;
```

Exemple : Boucles "uniques"

Les boucles dans l'exemple sont parcourues exactement une fois puisque valeur initiale = valeur finale. Et ce n'est pas important si l'incrément est positif ou négatif.

```
FOR i:= 10 TO 10 DO (* Boucle unique *)
  C:= C * COS(B) ;
END_FOR ;
```

ou

```
FOR i:= 10 TO 10 BY -1 DO (* Boucle unique *)
  C:= C * COS(B) ;
END_FOR ;
```

Exemple : Boucles critiques

Si l'incrément dans l'exemple est `j` > 0, alors les instructions sont exécutées.

Si `j` < est 0, les instructions ne sont pas exécutées car la situation Valeur initiale< Valeur finale ne permet qu'un incrément ≥ 0.

Si `j` = 0, les instructions sont exécutées et on obtient une boucle continue car avec un incrément de 0 la valeur finale n'est jamais atteinte.

```
FOR i:= 1 TO 10 BY j DO
  C:= C * COS(B) ;
END_FOR ;
```

Instruction de répétition WHILE...DO...END_WHILE

Description

L'instruction WHILE provoque l'exécution répétée d'une chaîne d'instructions jusqu'à ce que l'expression booléenne correspondante soit 0 (fausse). Si l'expression est fausse dès le départ, le groupe d'instructions n'est pas exécuté.

L'instruction DO marque la fin de définition d'une occurrence et le début d'une instruction (des instructions).

La répétition peut être prématurément quittée avec l'instruction EXIT.

L'instruction END_WHILE marque la fin de l'instruction (des instructions).

Dans les cas suivants WHILE ne doit pas être utilisé car cela forme une boucle continue qui entraîne un défaut bloquant du programme :

- WHILE ne doit pas servir à effectuer une synchronisation entre processus, par ex. une "boucle d'attente" avec une condition de fin définie en externe.
- WHILE ne doit pas être utilisé dans un algorithme pour lequel la satisfaction de la condition de fin de la boucle ou l'exécution d'une instruction EXIT ne peuvent pas être garanties.

Exemple WHILE...DO...END_WHILE

```
x := 1; WHILE x <= 100 DO x := x + 4; END_WHILE ;
```

Voir également

EXIT *(voir page 535)*

Instruction récurrente REPEAT...UNTIL...END_REPEAT

Description

L'instruction REPEAT provoque la répétition d'une chaîne d'instructions (au moins une fois) jusqu'à ce que la condition booléenne correspondante soit 1 (vraie).

L'instruction UNTIL marque la condition de fin.

La répétition peut être prématurément quittée avec l'instruction EXIT.

L'instruction END_REPEAT marque la fin de l'instruction (des instructions).

Dans les cas suivants REPEAT ne doit pas être utilisé car cela forme une boucle continue qui entraîne un défaut bloquant du programme :

- REPEAT ne doit pas servir à effectuer une synchronisation entre processus, par ex. une "boucle d'attente" avec une condition finale définie en externe.
- REPEAT ne doit pas être utilisé dans un algorithme pour lequel la satisfaction de la condition de fin de la boucle ou l'exécution d'une instruction EXIT ne peuvent pas être garanties.

Exemple REPEAT...UNTIL...END_REPEAT

```
x := -1
REPEAT
          x := x + 2
   UNTIL x >= 101
END_REPEAT ;
```

Voir également

EXIT *(voir page 535)*

Instruction récurrente EXIT

Description

L'instruction EXIT est utilisée pour terminer les instructions récurrentes (FOR, WHILE, REPEAT) avant que la condition de fin ne soit atteinte.

Si l'instruction EXIT se trouve dans une occurrence imbriquée, la boucle la plus interne (dans laquelle se trouve EXIT) est abandonnée. Et l'instruction suivante qui est exécutée est la première instruction après la fin de la boucle (END_FOR, END_WHILE ou END_REPEAT).

Exemple EXIT

Si FLAG a la valeur 0, SUM est 15 après exécution des instructions.

Si FLAG a la valeur 1, SUM est 6 après exécution des instructions.

```
SUM : = 0 ;
FOR I := 1 TO 3 DO
   FOR J := 1 TO 2 DO
      IF FLAG=1 THEN EXIT;
      END_IF ;
      SUM := SUM + J ;
   END_FOR ;
   SUM := SUM + I ;
END_FOR
```

Voir également

CASE *(voir page 529)*

WHILE *(voir page 533)*

REPEAT *(voir page 534)*

Appel de sous-programme

Appel de sous-programme

L'appel d'un sous-programme comprend le nom de la section du sous-programme suivi d'une liste de paramètres vide.

Les appels de sous-programmes ne fournissent pas de valeurs de retour.

Le sous-programme à appeler doit se trouver dans la même tâche que la section ST appelante.

Il est possible d'appeler des sous-programmes au sein de sous-programmes.

par ex.

```
Nom du sous-programme () ;
```

Les appels de sous-programme sont un complément de la norme CEI 61131-3 et doivent être activés de manière explicite.

Dans les sections d'actions SFC, les appels de sous-programmes ne sont autorisés que si le mode Multitoken a été activé.

RETURN

Description

Des instructions RETURN peuvent être utilisées dans des blocs de fonction dérivés DFB et des SR (sous-programmes).

Des instructions RETURN ne peuvent pas être utilisées dans le programme principal.

- Dans un DFB, une instruction RETURN force le retour au programme qui a appelé le DFB.
  - Le reste de la section de DFB contenant l'instruction RETURN n'est pas exécuté.
  - Les sections suivantes du DFB ne sont pas exécutées.

  Le programme qui a appelé le DFB sera exécuté après retour du DFB.
  Si le DFB est appelé par un autre DFB, le DFB appelant sera exécuté après le retour.
- Dans un SR, une instruction RETURN force le retour au programme qui a appelé le SR.
  - Le reste du SR contenant l'instruction RETURN n'est pas exécuté.

  Le programme qui a appelé le SR sera exécuté après retour du SR.

Instruction d'espacement

Description

Un seul point-virgule `;` représente une instruction d'espacement.

par ex.

```
IF x THEN ; ELSE ..
```

Dans l'exemple l'instruction qui suit l'instruction `THEN` est une instruction d'espacement. Cela signifie que le programme quitte l'instruction `IF` dès que la condition `IF` atteint 1.

Libellés et sauts

Présentation

Les étiquettes servent de cible à atteindre pour les sauts.

Les répères et les sauts en ST constituent une extension de la norme CEI 61131-3 et doivent être activés explicitement.

Propriétés des étiquettes

Propriétés des étiquettes :

- les étiquettes doivent toujours être le premier élément d'une ligne.
- les étiquettes ne peuvent se trouver que devant les instructions du premier ordre (pas dans les boucles).
- les étiquettes doivent être uniques dans toute la section, on ne fait pas de distinction ici entre majuscules et minuscules.
- la longueur maximale d'un étiquette est de 32 caractères.
- les étiquettes doivent satisfaire aux conventions de nom CEI générales.
- les étiquettes sont séparés par deux points `:` de la commande suivante.

Caractéristiques d'un saut

Caractéristiques d'un saut

- les sauts sont possibles au sein des sections de DFB et de programme.
- les sauts ne sont possibles qu'au sein des sections actuelles.

Exemple

```
IF var1 THEN
   JMP START;
:
:
START : ...
```

Commentaire

Description

Dans l'éditeur ST, les commentaires commencent par la chaîne de caractères `(*` et se terminent par la chaîne de caractères `*)`. Vous pouvez entrer un commentaire quelconque entre ces deux chaînes de caractères. Les commentaires peuvent être saisis à n'importe quelle position dans l'éditeur ST, à l'exception des mots clés, libellés, identificateurs et variables.

Selon la norme CEI 61131-3, il n'est pas possible d'imbriquer des commentaires. Toutefois, si des commentaires sont imbriqués, ils doivent être activés de manière explicite.

15.3 Appel de fonctions élémentaires, de blocs fonction élémentaires, de blocs fonction dérivés et de procédures

Objet de ce sous-chapitre

Appel de fonctions élémentaires, de blocs fonction élémentaires, de blocs fonction dérivés et de procédures dans le langage de programmation ST.

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Appel de fonctions élémentaires

Fonctions élémentaires

Les fonctions élémentaires sont disponibles sous forme de bibliothèques. La logique des fonctions est créée dans le langage de programmation C et ne peut pas être modifiée dans l'éditeur ST.

Les fonctions n'ont pas d'état interne. Lorsque les valeurs d'entrée sont identiques, la valeur de sortie est la même à chaque exécution de la fonction. Par exemple, l'addition de deux valeurs donne toujours le même résultat.

Certaines fonctions élémentaires peuvent être étendues à plus de deux entrées.

Les fonctions élémentaires n'ont qu'une seule valeur de renvoi (sortie).

Paramètres

Pour importer des valeurs dans une fonction ou exporter des valeurs d'une fonction, on a besoin d'« entrées » et d'une « sortie ». Ces entrées et cette sortie sont appelées « paramètres formels ».

Les états actuels du processus sont transmis aux paramètres formels. Ce sont les paramètres réels.

On peut utiliser un paramètre réel d'entrée de fonction de type :

- variable,
  adresse,
  valeur littérale.
  Expression ST

On peut utiliser un paramètre réel de sortie de fonction de type :

- variable,
- adresse.

Le type des données du paramètre réel doit correspondre au type des données du paramètre formel. La seule exception concerne les paramètres formels génériques dont le type de données est déterminé par le paramètre réel.

De plus, pour les paramètres formels génériques du type de données `ANY_BIT`, des paramètres réels du type de données `INT` ou `DINT` (pas `UINT` ni `UDINT`) peuvent être utilisés.

Il s'agit d'une extension de la norme CEI 61131-3, qui doit être activée de manière explicite.

Exemple :

Autorisé :

```
AND (AnyBitParam := IntVar1, AnyBitParam2 := IntVar2);
```

Non autorisé :

```
AND_WORD (WordParam1 := IntVar1, WordParam2 := IntVar2);
```

(Dans ce cas AND_INT doit être utilisé.)

```
AND_ARRAY_WORD (ArrayInt, ...);
```

(Dans ce cas, un changement de type explicite doit être effectué via INT_ARR_TO_WORD_ARR (...);.

Il n'est en principe pas nécessaire d'affecter une valeur à tous les paramètres formels. Pour connaître les types de paramètre formel pour lesquels cela est cependant impératif, veuillez vous reporter au tableau.

Type de paramètre	EDT	STRING	ARRAY	ANY_ARRAY	IODDT	STRUCT	FB	ANY
Entrée	-	-	+	+	+	+	+	+
VAR_IN_OUT	+	+	+	+	+	+	/	+
Sortie	-	-	-	-	-	-	/	-
+ paramètre réel impératif								
- paramètre réel non impératif								
/ non applicable								

Si aucune valeur n'est affectée à un paramètre formel, la valeur initiale est utilisée pendant l'exécution du bloc fonction. Si aucune valeur initiale n'est définie, la valeur par défaut (0) est utilisée.

Remarques sur la programmation

Veuillez tenir compte des remarques qui suivent sur la programmation :

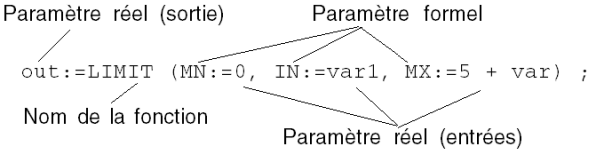
- Toutes les fonctions génériques sont surchargées. Cela signifie que les fonctions peuvent être appelées avec ou sans la saisie du type de données.
  Ex. :
  ```
  i1 := ADD (i2, 3);
  ```
  est identique à
  ```
  i1 := ADD_INT (i2, 3);
  ```
- Les fonctions peuvent être imbriquées (voir également *Imbrication de fonctions, page 546*).
- Les fonctions ne sont exécutées que lorsque l'entrée EN = 1 ou lorsque l'entrée EN est désactivée (voir aussi *EN et ENO, page 547*).
- Il existe deux façons d'appeler une fonction :
  - appel formel (appel d'une fonction avec les noms des paramètres formels),
  - appel informel (appel d'une fonction sans les noms des paramètres formels).

Appel formel

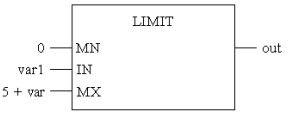
Lors d'un appel formel (avec les noms des paramètres formels), l'appel se compose du paramètre réel de la sortie, suivi de l'instruction d'affectation :=, suivie du nom de fonction, suivi d'une liste entre parenthèses des affectations de valeur (paramètres réels) aux paramètres formels. L'ordre d'énumération des paramètres formels dans l'appel d'une fonction **n'est pas important**.

Il est possible d'utiliser EN et ENO avec ce type d'appel.

Appel d'une fonction avec les noms des paramètres formels :



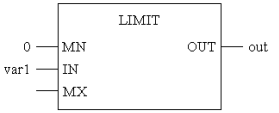
Appel de la même fonction dans FBD :



Lors d'un appel formel, il n'est pas nécessaire d'affecter une valeur à tous les paramètres formels (voir également *Paramètres, page 542*).

```
out:=LIMIT (MN:=0, IN:=var1) ;
```

Appel de la même fonction dans FBD :

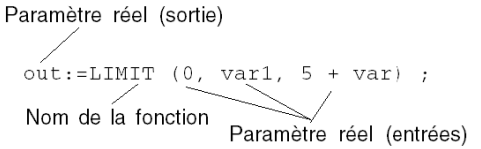


Appel informel

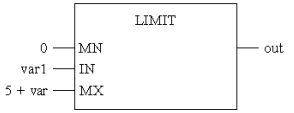
Lors d'un appel informel (sans les noms des paramètres formels), l'appel se compose du paramètre réel de la sortie, suivi du symbole de l'instruction d'affectation `:=`, suivi du nom de fonction, suivi d'une liste entre parenthèses des paramètres réels des entrées. L'ordre d'énumération des paramètres réels dans l'appel d'une fonction **est important**.

`EN` et `ENO` ne peuvent **pas** être utilisés avec ce type d'appel.

Appel d'une fonction sans les noms des paramètres formels :



Appel de la même fonction dans FBD :



Même lors d'un appel informel, il n'est pas nécessaire d'affecter une valeur à tous les paramètres formels (voir également *Paramètres, page 542*).

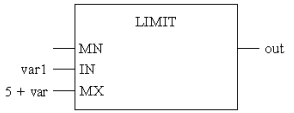
Il s'agit d'une extension de la norme CEI 61131-3, qui doit être activée de manière explicite.

Un champ de paramètre vide permet d'omettre un paramètre.

Appel avec un champ de paramètre vide :

```
out:=LIMIT ( ,var1, 5 + var) ;
```

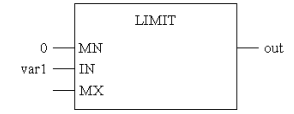
Appel de la même fonction dans FBD :



Si les paramètres formels sont omis à la fin, aucun champ de paramètre vide ne doit être utilisé.

```
out:=LIMIT (0, var1) ;
```

Appel de la même fonction dans FBD :



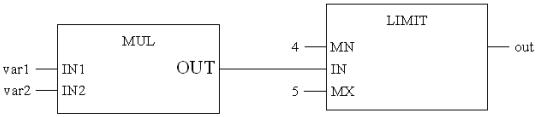
Imbrication de fonctions

L'appel d'une fonction peut contenir l'appel d'autres fonctions. Le niveau d'imbrication est illimité.

Appel d'une fonction imbriquée :

```
out:=LIMIT (MN:=4, IN:=MUL(IN1:=var1, IN2:=var2), MX:=5) ;
```

Appel de la même fonction dans FBD :



Les fonctions qui livrent une valeur du type de données `ANY_ARRAY` **ne peuvent pas** être utilisées **au sein** d'un appel de fonction.

Imbrication **non autorisée** avec `ANY_ARRAY` :

```
                    ANY_ARRAY
                    /
out:=LIMIT (MN:=4,  IN:=EXAMP(IN1:=var1,  IN2:=var2),  MX:=5) ;
```

Les types de données ANY_ARRAY sont permis comme valeur de renvoi de la fonction appelant ou comme paramètre des fonctions imbriquées.

Imbrication **autorisée** avec `ANY_ARRAY` :

```
ANY_ARRAY                    ANY_ARRAY                    ANY_ARRAY
   /                            /                            /
out:=EXAMP(MN:=4, IN:=EXAMP(IN1:=var1, IN2:=var2), MX:=var3)
```

EN et ENO

Pour toutes les fonctions, une entrée EN et une sortie ENO peuvent être configurées.

Si la valeur d'EN est égale à « 0 », lorsque la fonction est appelée, les algorithmes définis par cette dernière ne sont pas exécutés et ENO est mis sur « 0 ».

Si la valeur d'EN est égale à « 1 », lorsque la fonction est appelée, les algorithmes définis par la fonction sont exécutés. Après l'exécution exempte d'erreur de ces algorithmes, la valeur d'ENO est mise sur « 1 ». Si une erreur se produit durant l'exécution de ces algorithmes, ENO est mis sur « 0 ».

Si aucune valeur n'est attribuée à la broche EN à l'appel du FFB, l'algorithme défini par ce dernier est exécuté (comme lorsque EN a la valeur « 1 »).

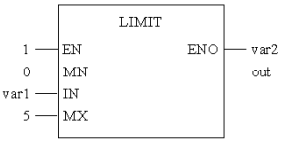
Si ENO est mis sur « 0 » (en raison de EN=0 ou d'une erreur d'exécution), la sortie de la fonction est mise sur « 0 ».

Le comportement de sortie de la fonction ne dépend pas de l'appel de la fonction sans EN/ENO ou avec EN=1.

Si EN/ENO doivent être utilisés, l'appel de la fonction doit être exécuté comme un appel formel.

```
out:=LIMIT (EN:=1, MN:=0, IN:=var1, MX:=5, ENO=>var2) ;
```

Appel de la même fonction dans FBD :



Bloc fonction élémentaire d'appel et bloc fonction dérivé

Bloc fonction élémentaire

Les blocs fonction élémentaires ont des états internes. Pour des valeurs égales aux entrées, la valeur à la sortie peut être différente à chaque exécution du bloc fonction. Pour un compteur, par exemple, la valeur de la sortie est incrémentée.

Les blocs fonction peuvent comprendre plusieurs valeurs de renvoi (sorties).

Bloc fonction dérivé

Les blocs fonction dérivés (DFB) ont les mêmes caractéristiques que les blocs fonction élémentaires. Ils sont cependant créés par l'utilisateur dans les langages FBD, LD, IL et/ou ST.

Paramètres

Pour importer des valeurs dans un bloc fonction ou les exporter d'un bloc fonction, des entrées et des sorties sont nécessaires. Ces entrées et ces sorties sont appelées « paramètres formels ».

Les états actuels du processus sont transmis aux paramètres formels. Ces états sont appelés « paramètres réels ».

Pour les entrées de bloc fonction, on peut utiliser un paramètre réel de type :
- variable,
- adresse,
- valeur littérale.

Pour les sorties de bloc fonction, on peut utiliser un paramètre réel de type :
- variable,
- adresse.

Le type des données du paramètre réel doit correspondre au type des données du paramètre formel. La seule exception concerne les paramètres formels génériques dont le type de données est déterminé par le paramètre réel.

De plus, pour les paramètres formels génériques du type de données `ANY_BIT`, des paramètres réels du type de données `INT` ou `DINT` (pas `UINT` ni `UDINT`) peuvent être utilisés.

Il s'agit d'une extension de la norme CEI 61131-3, qui doit être activée de manière explicite.

Exemple :

Autorisé :

```
AND (AnyBitParam := IntVar1, AnyBitParam2 := IntVar2);
```

Non autorisé :

```
AND_WORD (WordParam1 := IntVar1, WordParam2 := IntVar2);
```

(Dans ce cas `AND_INT` doit être utilisé.)

```
AND_ARRAY_WORD (ArrayInt, ...);
```

(Dans ce cas, un changement de type explicite doit être effectué via `INT_ARR_TO_WORD_ARR (...);`.)

Il n'est en principe pas nécessaire d'affecter une valeur à tous les paramètres formels. Pour connaître les types de paramètre formel pour lesquels cela est cependant impératif, veuillez vous reporter au tableau.

Type de paramètre	EDT	STRING	ARRAY	ANY_ARRAY	IODDT	STRUCT	FB	ANY
EFB : Entrée	-	+	+	+	/	+	/	+
EFB : VAR_IN_OUT	+	+	+	+	+	+	/	+
EFB : Sortie	-	-	+	+	+	-	/	+
DFB : Entrée	-	+	+	+	/	+	/	+
DFB : VAR_IN_OUT	+	+	+	+	+	+	/	+
DFB : Sortie	-	-	+	/	/	-	/	+
+ paramètre réel impératif								
- paramètre réel non impératif								
/ non applicable								

Si aucune valeur n'est affectée à un paramètre formel, la valeur initiale est utilisée pendant l'exécution du bloc fonction. Si aucune valeur initiale n'est définie, la valeur par défaut (0) est utilisée.

Si aucune valeur n'est affectée à un paramètre formel et que le bloc fonction/DFB ait été instancié à plusieurs reprises, les instances appelées par la suite travaillent avec l'ancienne valeur.

Variables publiques

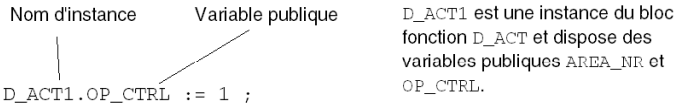
Certains blocs fonction disposent non seulement d'entrées et de sorties, mais également de variables publiques.

Ces variables permettent de transmettre des valeurs statistiques (valeurs non influencées par le procédé) au bloc fonction. Elles sont donc utilisées lors du paramétrage du bloc fonction.

Les variables publiques sont une extension de la norme CEI 61131-3.

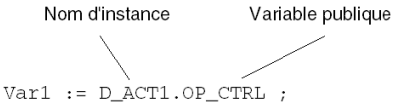
Des valeurs sont affectées aux variables publiques via leur valeur initiale ou via l'opérateur d'affectation.

Exemple :



Les valeurs des variables publiques sont ensuite lues à partir du nom d'instance du bloc fonction et du nom de la variable publique.

Exemple :



Variables privées

Certains blocs fonction disposent non seulement d'entrées, de sorties et de variables publiques, mais également de variables privées.

A l'instar des variables publiques, ces variables permettent de transmettre des valeurs statistiques (valeurs non influencées par le procédé) au bloc fonction.

Le programme utilisateur ne peut pas accéder à ces variables. Seule la table d'animation en a la capacité.

NOTE : les DFB imbriqués sont déclarés comme des variables privées du DFB parent. Ainsi, leurs variables ne sont pas accessibles via la programmation, mais via la table d'animation.

Les variables privées sont une extension de la norme CEI 61131-3.

Remarques sur la programmation

Veuillez tenir compte des remarques qui suivent sur la programmation :

- Les blocs fonction ne sont exécutés que lorsque l'entrée `EN` = 1 ou lorsque l'entrée `EN` est désactivée (voir aussi *EN et ENO, page 554*).
- L'affectation de variables aux sorties de type `ANY` ou `ARRAY` doit être effectuée via l'opérateur `=>` (voir aussi *Appel formel, page 551*).
  Une affectation en dehors de l'appel d'un bloc fonction n'est pas possible.
  L'instruction
  ```
  My_Var := My_SAH.OUT;
  ```
  n'est pas valide, la sortie `OUT` du bloc fonction `SAH` étant de type `ANY`.
  L'instruction
  ```
  Cal My_SAH (OUT=>My_Var);
  ```
  est en revanche **valide**.
- Des conditions particulières s'appliquent lors de l'utilisation de variables VAR_IN_OUT *(voir page 555)*.

- L'utilisation des blocs fonction comprend deux parties dans ST :
  - la déclaration *(voir page 551)*
  - l'appel du bloc fonction
- Il existe deux façons d'appeler un bloc fonction :
  - appel formel *(voir page 551)* (appel avec les noms des paramètres formels) Des variables peuvent ainsi être affectées aux sorties via l'opérateur =>
  - appel informel *(voir page 552)* (appel sans les noms des paramètres formels)
- Les instances de bloc fonction/DFB peuvent être appelées plusieurs fois, à l'exception des instances d'EFB de communication qui ne peuvent être appelées qu'une seule fois (voir *Appel multiple d'une instance de bloc fonction, page 554*).

Déclaration

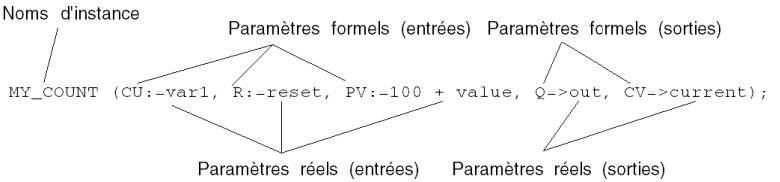
Avant d'être appelé, un bloc fonction doit être déclaré dans l'éditeur de variables.

Appel formel

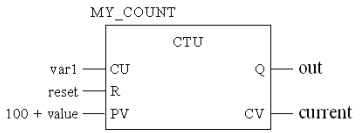
Pour l'appel formel (avec les noms des paramètres formels), les blocs fonction sont appelés via une instruction qui comprend le nom d'instance du bloc fonction suivi d'une liste entre parenthèses des affectations des paramètres réels aux paramètres formels. Affectez des paramètres formels d'entrée à l'aide de l'affectation := et des paramètres formels d'entrée à l'aide de l'affectation :=. L'ordre d'énumération des paramètres formels d'entrées et de sorties **n'est pas important**.

Il est possible d'utiliser EN et ENO avec ce type d'appel.

Appel d'un bloc fonction avec les noms des paramètres formels :



Appel du même bloc fonction dans FBD :



L'affectation de la valeur à une sortie de bloc fonction est réalisée si vous entrez le nom réel du paramètre, suivi de l'instruction d'affectation := suivie du nom de l'instance du bloc fonction et si vous chargez le paramètre formel de la sortie du bloc fonction (séparé par un point final).

Ex. :

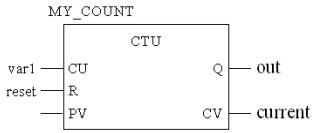
```
MY_COUNT (CU:=var1, R:=reset, PV:=100 + value); Q :=
MY_COUNT.out ; CV := MY_COUNT.current ;
```

NOTE : les DDT de type Array ne peuvent pas être affectés de cette manière, contrairement aux DDT de type Structure.

Il n'est pas nécessaire d'affecter une valeur à tous les paramètres formels (voir aussi *Paramètres, page 548*).

```
MY_COUNT (CU:=var1, R:=reset, Q=>out, CV=>current);
```

Appel du même bloc fonction dans FBD :

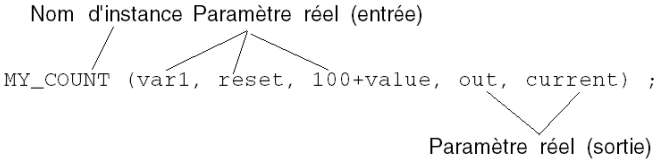


Appel informel

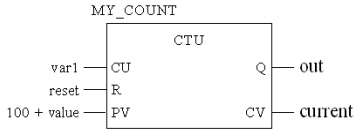
Pour l'appel informel (sans les noms des paramètres formels), les blocs fonction sont appelés via une instruction qui comprend le nom d'instance du bloc fonction suivi d'une liste entre parenthèses des paramètres réels des entrées et des sorties. L'ordre d'énumération des paramètres réels dans l'appel d'un bloc fonction **est important**.

`EN` et `ENO` ne peuvent **pas** être utilisés avec ce type d'appel.

Appel d'un bloc fonction sans les noms des paramètres formels :



Appel du même bloc fonction dans FBD :



Même lors d'un appel informel, il n'est pas nécessaire d'affecter une valeur à tous les paramètres formels (voir également *Paramètres, page 548*). Cela ne s'applique pas aux variables `VAR_IN_OUT`, aux paramètres d'entrée avec des longueurs dynamiques et aux sorties de type `ANY`. Une variable doit toujours être affectée.

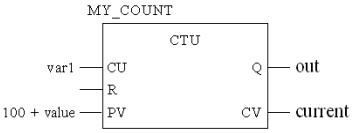
Il s'agit d'une extension de la norme CEI 61131-3, qui doit être activée de manière explicite.

Un champ de paramètre vide permet d'omettre un paramètre.

Appel avec un champ de paramètre vide :

```
MY_COUNT (var1, , 100 + value, out, current) ;
```

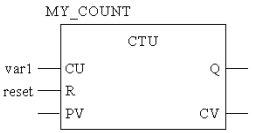
Appel du même bloc fonction dans FBD :



Si les paramètres formels sont omis à la fin, aucun champ de paramètre vide ne doit être utilisé.

```
MY_COUNT (var1, reset) ;
```

Appel du même bloc fonction dans FBD :



Appel d'un bloc fonction sans entrées

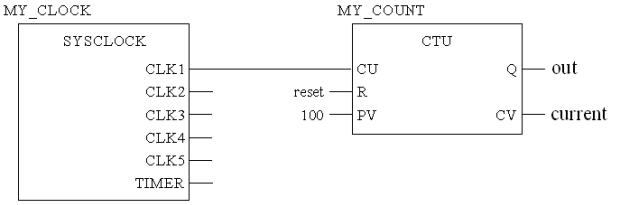
Même si le bloc fonction ne dispose pas d'entrées ou si les entrées ne sont pas à paramétrer, vous devez appeler le bloc fonction avant que ses sorties puissent être utilisées. Faute de quoi, le système transmettra les valeurs initiales des sorties, c'est-à-dire « 0 ».

Ex. :

Appel du bloc fonction dans ST :

```
MY_CLOCK () ;MY_COUNT (CU:=MY_CLOCK.CLK1, R:=reset, PV:=100,
Q=>out, CV=>current) ;
```

Appel du même bloc fonction dans FBD :



Appel multiple d'une instance de bloc fonction

Les instances de bloc fonction/DFB peuvent être appelées plusieurs fois, à l'exception des instances d'EFB de communication qui ne peuvent être appelées qu'une seule fois.

L'appel multiple de la même instance de DFB/bloc fonction est par exemple utile dans les cas suivants :

- si le bloc fonction/DFB ne comporte aucune valeur interne ou si celle-ci n'est plus nécessaire pour un traitement ultérieur.
  Dans ce cas, l'appel multiple de la même instance de DFB/bloc fonction permet d'économiser de l'espace mémoire, car le code du bloc fonction/DFB n'est alors chargé qu'une seule fois.
  Le bloc fonction/DFB est pour ainsi dire traité comme une « fonction ».
- si le bloc fonction/DFB comprend une valeur interne, qui doit influencer différents endroits du programme (la valeur d'un compteur, par exemple, doit être augmentée dans différentes parties du programme).
  Dans ce cas, l'appel multiple de la même instance de bloc fonction/DFB permet d'économiser la mémoire des résultats intermédiaires pour un traitement ultérieur à un autre endroit du programme.

EN et ENO

Pour tous les blocs fonction/DFB, une entrée EN et une sortie ENO peuvent être configurées.

Au cas où la valeur d'EN est égale à « 0 », lorsque le bloc fonction/DFB est appelé, les algorithmes définis par ce dernier ne sont pas exécutés et ENO est mis à « 0 ».

Au cas où la valeur d'EN est égale à « 1 », lorsque le bloc fonction/DFB est appelé, les algorithmes définis par ce dernier sont exécutés. Une fois les algorithmes exécutés, la valeur de la sortie ENO est réglée sur « 1 ». Si une erreur se produit durant l'exécution de ces algorithmes, ENO est mis à « 0 ».

Si aucune valeur n'est attribuée à la broche EN à l'appel du FFB, l'algorithme défini par ce dernier est exécuté (comme lorsque EN a la valeur « 1 »).

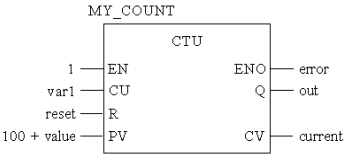
Si ENO est mis à « 0 » (du fait de EN = 0 ou d'une erreur d'exécution), les sorties du bloc fonction/DFB conservent l'état qu'elles avaient au dernier cycle exécuté correctement.

Le comportement aux sorties des blocs fonction/DFB est le même, que les blocs fonction/DFB aient été appelés sans EN/ENO ou avec EN = 1.

Si EN/ENO doivent être utilisés, l'appel du bloc fonction doit être exécuté sous forme d'appel formel. L'affectation d'une variable à ENO doit être effectuée avec l'opérateur =>.

```
MY_COUNT (EN:=1, CU:=var1, R:=reset, PV:=100 + value,
ENO=>error, Q=>out, CV=>current) ;
```

Appel du même bloc fonction dans FBD :



Variable VAR_IN_OUT

Très souvent, on utilise des blocs fonction pour lire une variable au niveau de l'entrée (variables d'entrée), traiter celle-ci, et sortir à nouveau les valeurs modifiées de la même variable (variables de sortie). Ce cas particulier d'une variable d'entrée/de sortie est également appelé variable VAR_IN_OUT.

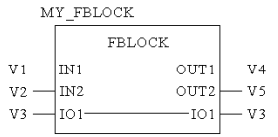
Il convient de noter les particularités suivantes lors de l'utilisation de blocs fonction/DFB avec des variables VAR_IN_OUT :

- une variable doit être affectée à toutes les entrées VAR_IN_OUT.
- il est interdit d'affecter des valeurs littérales ou des constantes aux entrées VAR_IN_OUT.
- **aucune** valeur ne doit être affectée aux sorties VAR_IN_OUT.
- les variables VAR_IN_OUT ne peuvent **pas** être utilisées en dehors de l'appel du bloc fonction.

Appel d'un bloc fonction avec une variable VAR_IN_OUT dans ST :

```
MY_FBLOCK(IN1:=V1, IN2:=V2, IO1:=V3, OUT1=>V4, OUT2=>V5);
```

Appel du même bloc fonction dans FBD :



Les variables VAR_IN_OUT ne peuvent **pas** être utilisées en dehors de l'appel du bloc fonction.

Les appels de blocs fonction suivants sont par conséquent **non valides** :

Appel **non valide**, exemple 1 :

`InOutFB.inout := V1;`	Affectation des variables `V1` à un paramètre `VAR_IN_OUT`. **Erreur :** l'opération ne peut pas être exécutée, car il n'est pas possible d'accéder au paramètre `VAR_IN_OUT` en dehors de l'appel d'un bloc fonction.

Appel **non valide**, exemple 2 :

`V1 := InOutFB.inout;`	Affectation d'un paramètre `VAR_IN_OUT` à la variable `V1`. **Erreur :** l'opération ne peut pas être exécutée, car il n'est pas possible d'accéder au paramètre `VAR_IN_OUT` en dehors de l'appel d'un bloc fonction.

Les appels de blocs fonction suivants sont en revanche **valides** :

Appel **valide**, exemple 1 :

`InOutFB (inout:=V1);`	Appel d'un bloc fonction avec un paramètre `VAR_IN_OUT` et affectation formelle des paramètres réels au sein de l'appel de bloc fonction.

Appel **valide**, exemple 2 :

`InOutFB (V1);`	Appel d'un bloc fonction avec un paramètre `VAR_IN_OUT` et affectation informelle des paramètres réels au sein de l'appel de bloc fonction.

Procédures

Procédure

Les procédures sont disponibles sous forme de bibliothèques. La logique des procédures est créée dans le langage de programmation C et ne peut pas être modifiée dans l'éditeur ST.

Comme les fonctions, les procédures n'ont pas d'états internes. Lorsque les valeurs d'entrée sont identiques, la valeur de sortie est la même à chaque exécution de la procédure. Par exemple, l'addition de deux valeurs donne toujours le même résultat.

Contrairement aux fonctions, les procédures ne livrent aucune valeur de renvoi et prennent en charge les variables `VAR_IN_OUT`.

Les procédures sont un complément de la norme CEI 61131-3 et doivent être activées de manière explicite.

Paramètres

Pour importer des valeurs dans une procédure ou exporter des valeurs d'une procédure, on a besoin d'« entrées » et de « sorties ». Ces entrées et ces sorties sont appelées « paramètres formels ».

Les états actuels du processus sont transmis aux paramètres formels. Ce sont les paramètres réels.

On peut utiliser comme paramètre réel des entrées de procédure :
- variable,
- adresse,
- valeur littérale.
- Expression ST

On peut utiliser comme paramètre réel des sorties de procédure :
- variable,
- adresse.

Le type des données du paramètre réel doit correspondre au type des données du paramètre formel. La seule exception concerne les paramètres formels génériques dont le type de données est déterminé par le paramètre réel.

De plus, pour les paramètres formels génériques du type de données `ANY_BIT`, des paramètres réels du type de données `INT` ou `DINT` (pas `UINT` ni `UDINT`) peuvent être utilisés.

Il s'agit d'une extension de la norme CEI 61131-3, qui doit être activée de manière explicite.

Exemple :

Autorisé :

```
AND (AnyBitParam := IntVar1, AnyBitParam2 := IntVar2);
```

Non autorisé :

```
AND_WORD (WordParam1 := IntVar1, WordParam2 := IntVar2);
```

(Dans ce cas `AND_INT` doit être utilisé.)

```
AND_ARRAY_WORD (ArrayInt, ...);
```

(Dans ce cas, un changement de type explicite doit être effectué via `INT_ARR_TO_WORD_ARR (...);`.

Il n'est en principe pas nécessaire d'affecter une valeur à tous les paramètres formels. Pour connaître les types de paramètre formel pour lesquels cela est cependant impératif, veuillez vous reporter au tableau.

Type de paramètre	EDT	STRING	ARRAY	ANY_ARRAY	IODDT	STRUCT	FB	ANY
Entrée	-	-	+	+	+	+	+	+
VAR_IN_OUT	+	+	+	+	+	+	/	+
Sortie	-	-	-	-	-	-	/	+
+ paramètre réel impératif								
- paramètre réel non impératif								
/ non applicable								

Si aucune valeur n'est affectée à un paramètre formel, la valeur initiale est utilisée pendant l'exécution du bloc fonction. Si aucune valeur initiale n'est définie, la valeur par défaut (0) est utilisée.

Remarques sur la programmation

Veuillez tenir compte des remarques qui suivent sur la programmation :

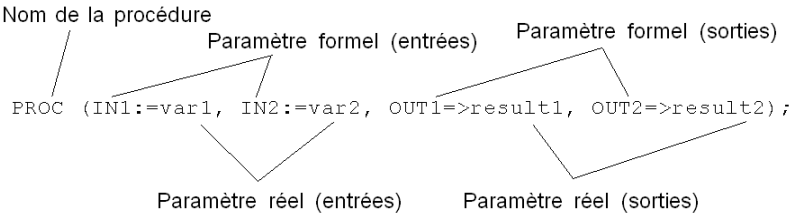
- Les procédures ne sont exécutées que lorsque l'entrée `EN` = 1 ou lorsque l'entrée `EN` est désactivée (voir aussi *EN et ENO, page 561*).
- Des conditions particulières s'appliquent lors de l'utilisation de variables VAR_IN_OUT *(voir page 561)*.
- Il existe deux façons d'appeler une procédure :
  - appel formel *(voir page 559)* (appel avec les noms des paramètres formels) Des variables peuvent ainsi être affectées aux sorties via l'opérateur `=>`
  - appel informel *(voir page 560)* (appel sans les noms des paramètres formels)

Appel formel

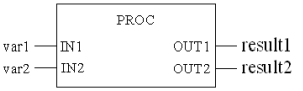
Pour l'appel formel (avec les noms des paramètres formels), les procédures sont appelées via une instruction qui comprend le nom de la procédure suivi d'une liste entre parenthèses des affectations des paramètres réels aux paramètres formels. L'affectation des paramètres formels des entrées s'effectue via l'affectation := et l'affectation des paramètres formels des sorties via l'affectation =>. L'ordre d'énumération des paramètres formels d'entrées et de sorties **n'est pas important**.

Il est possible d'utiliser `EN` et `ENO` avec ce type d'appel.

Appel d'une procédure avec les noms des paramètres formels :



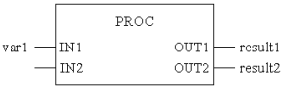
Appel de la même procédure dans FBD :



Lors d'un appel formel, il n'est pas nécessaire d'affecter une valeur à tous les paramètres formels (voir également *Paramètres, page 557*).

```
PROC (IN1:=var1, OUT1=>result1, OUT2=>result2);
```

Appel de la même procédure dans FBD :

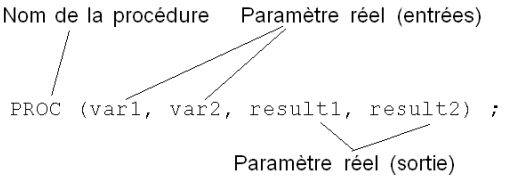


Appel informel

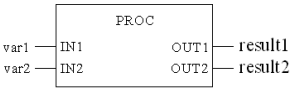
Pour l'appel informel (sans les noms des paramètres formels), les procédures sont appelées via une instruction qui comprend le nom de la procédure suivi d'une liste entre parenthèses des paramètres réels des entrées et des sorties. L'ordre d'énumération des paramètres réels dans l'appel d'une procédure **est important**.

`EN` et `ENO` ne peuvent **pas** être utilisés avec ce type d'appel.

Appel d'une procédure sans les noms des paramètres formels :



Appel de la même procédure dans FBD :



Même lors d'un appel informel, il n'est pas nécessaire d'affecter une valeur à tous les paramètres formels (voir également *Paramètres, page 557*).

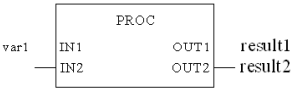
Il s'agit d'une extension de la norme CEI 61131-3, qui doit être activée de manière explicite.

Un champ de paramètre vide permet d'omettre un paramètre.

Appel avec un champ de paramètre vide :

```
PROC (var1, , result1, result2) ;
```

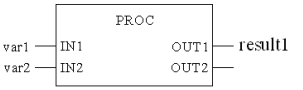
Appel de la même procédure dans FBD :



Si les paramètres formels sont omis à la fin, aucun champ de paramètre vide ne doit être utilisé.

```
PROC (var1, var2, result1) ;
```

Appel de la même procédure dans FBD :



EN et ENO

Pour toutes les procédures, une entrée EN et une sortie ENO peuvent être configurées.

Si la valeur d'EN est égale à « 0 », lorsque la procédure est appelée, les algorithmes définis par cette dernière ne sont pas exécutés et ENO est mis sur « 0 ».

Si la valeur d'EN est égale à « 1 », lorsque la procédure est appelée, les algorithmes définis par la procédure sont exécutés. Après l'exécution exempte d'erreur de ces algorithmes, la valeur d'ENO est mise sur « 1 ». Si une erreur se produit durant l'exécution de ces algorithmes, ENO est mis sur « 0 ».

Si aucune valeur n'est attribuée à la broche EN à l'appel du FFB, l'algorithme défini par ce dernier est exécuté (comme lorsque EN a la valeur « 1 »).

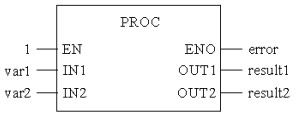
Si ENO est mis sur « 0 » (en raison de EN=0 ou d'une erreur d'exécution), les sorties de la procédure sont mises sur « 0 ».

Le comportement de la procédure à la sortie ne dépend pas du fait que la procédure ait été appelée sans EN ou avec EN=1.

Si EN/ENO doivent être utilisés, l'appel de la procédure doit être exécuté comme un appel formel. L'affectation d'une variable à ENO doit être effectuée avec l'opérateur =>.

```
PROC (EN:=1, IN1:=var1, IN2:=var2, ENO=>error,
OUT1=>result1, OUT2=>result2) ;
```

Appel de la même procédure dans FBD :



Variable VAR_IN_OUT

Très souvent, on utilise des procédures pour lire une variable au niveau de l'entrée (variables d'entrée), traiter celle-ci, et sortir à nouveau les valeurs modifiées de la même variable (variables de sortie). Ce cas particulier d'une variable d'entrée/de sortie est également appelé variable VAR_IN_OUT.

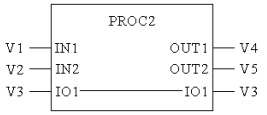
Il convient de noter les particularités suivantes dans le cas de l'utilisation de procédures avec des variables VAR_IN_OUT :

- une variable doit être affectée à toutes les entrées VAR_IN_OUT.
- il est interdit d'affecter des valeurs littérales ou des constantes aux entrées VAR_IN_OUT.
- **aucune** valeur ne doit être affectée aux sorties VAR_IN_OUT.
- les variables VAR_IN_OUT ne peuvent **pas** être utilisées en dehors de l'appel de procédure.

Appel d'une procédure avec une variable VAR_IN_OUT dans ST :

```
PROC2 (IN1:=V1, IN2:=V2, IO1:=V3, OUT1=>V4, OUT2=>V5) ;
```

Appel de la même procédure dans FBD :



Les variables VAR_IN_OUT ne peuvent **pas** être utilisées en dehors de l'appel de procédure.

Les appels de procédure suivants sont par conséquent **invalides** :

Appel **non valide**, exemple 1 :

`InOutProc.inout := V1;`	Affectation des variables V1 à un paramètre VAR_IN_OUT. **Erreur :** l'opération ne peut pas être exécutée, car il n'est pas possible d'accéder au paramètre VAR_IN_OUT en dehors de l'appel de procédure.

Appel **non valide**, exemple 2 :

`V1 := InOutProc.inout;`	Affectation d'un paramètre VAR_IN_OUT à la variable V1. **Erreur :** l'opération ne peut pas être exécutée, car il n'est pas possible d'accéder au paramètre VAR_IN_OUT en dehors de l'appel de procédure.

Les appels de procédure suivants sont en revanche **valides** :

Appel **valide**, exemple 1 :

`InOutProc (inout:=V1);`	Appel d'une procédure avec un paramètre VAR_IN_OUT et affectation formelle des paramètres réels au sein de l'appel de procédure.

Appel **valide**, exemple 2 :

`InOutProc (V1);`	Appel d'une procédure avec un paramètre VAR_IN_OUT et affectation informelle des paramètres réels au sein de l'appel de procédure.

Blocs fonction utilisateur (DFB)

V

Dans cette partie

Cette partie présente :

- les blocs fonctions utilisateur (DFB),
- la structure interne des DFB,
- les DFB de diagnostic,
- les types et instances des DFB,
- les appels d'instances à partir de différents langages.

Contenu de cette partie

Cette partie contient les chapitres suivants :

Présentation de Blocs Fonction Utilisateur (DFB)

16

Objet de ce chapitre

Ce chapitre présente les blocs fonction utilisateur (DFB), et les différentes étapes de mise en oeuvre.

Contenu de ce chapitre

Ce chapitre contient les sujets suivants :

Présentation des blocs fonctions utilisateur (DFB)

Présentation

Le logiciel Unity Pro vous permet de créer des blocs fonction utilisateur DFB, en utilisant les langages d'automatismes. Un DFB est un bloc de programme que vous avez écrit, afin de répondre aux spécificités de votre application. Il comprend:

- une ou plusieurs sections écrites en langage à contacts (LD), en liste d'instructions (IL), en littéral structuré (ST) ou en langage à blocs fonctionnels (FBD),
- des paramètres d'entrée/de sortie,
- des variables internes publiques ou privées.

Les blocs fonction vous permettent de structurer et d'optimiser votre application. Vous pouvez les utiliser dès qu'une séquence de programme est répétée plusieurs fois dans votre application ou pour figer une programmation standard (par exemple, l'algorithme de commande d'un moteur incluant la prise en compte des sécurités locales).

L'export puis l'import de ces blocs fonction permet leur utilisation par un groupe de programmeurs travaillant sur une même application ou dans des applications différentes.

Avantage d'utiliser un DFB

L'utilisation d'un bloc fonction DFB dans une application vous permet :

- de simplifier la conception et la saisie du programme,
- d'accroître la lisibilité du programme,
- de faciliter la mise au point de l'application (toutes les variables manipulées par le bloc fonction sont identifiées sur son interface),
- de diminuer le volume de code généré (le code correspondant au DFB est chargé une seule fois, quel que soit le nombre d'appels au DFB dans le programme, seules les données correspondants aux instances sont générées).

Comparaison avec un sous-programme

Par rapport à un sous programme, l'utilisation d'un DFB permet :

- de paramétrer plus facilement le traitement,
- d'utiliser des variables internes propres au DFB, donc indépendantes de l'application,
- de tester son fonctionnement indépendamment de l'application.

De plus, les langages LD et FBD permettent de visualiser de manière graphique les DFB, ce qui facilite la conception et la mise au point de votre programme.

DFB créés avec les logiciels précédents

Les DFB créés avec PL7 et Concept doivent être au préalable convertis à l'aide des convertisseurs inclus dans le produit avant d'être utilisés dans l'application.

Domaine d'utilisation

Le tableau ci-après décrit le domaine d'utilisation des DFB.

Fonction	Domaine
Automates pour lesquels les DFB sont utilisables.	Premium\Atrium et Quantum
Logiciel de création des DFB	Unity Pro
Logiciels avec lesquels les DFB sont utilisables.	Unity Pro ou Unity Pro Medium
Langage de programmation pour la création du code des DFB.	IL, ST, LD ou FBD (1)
Langage de programmation avec lesquels les DFB sont utilisables.	IL, ST, LD ou FBD (1)

(1) IL: Liste d'Instructions, ST: littéral STructuré, LD: Langage à contacts (LaDder), FBD: langage à Blocs Fonctionnels.

Mise en oeuvre d'un bloc fonction DFB

Procédure de mise en œuvre

La procédure de mise en œuvre du bloc fonction DFB comporte trois étapes :

Etape	Action
1	Création de votre modèle DFB (appelé : type DFB).
2	Création d'une copie de ce bloc fonction, appelé une instance, à chaque fois que le DFB est utilisé dans l'application.
3	Utilisation des instances DFB dans votre programme d'application.

Création du type DFB

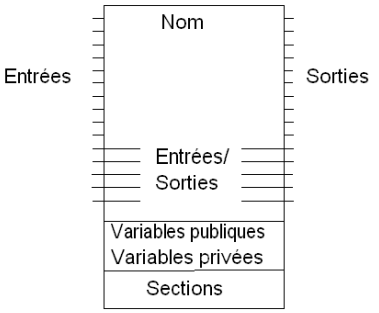
Cette opération consiste à concevoir un modèle du DFB que vous souhaitez utiliser dans votre application. Pour ce faire, utilisez l'éditeur DFB pour définir et coder tous les éléments qui constituent le DFB :

- Description du bloc fonction : nom, type (DFB), activation du diagnostic, commentaire.
- Structure du bloc fonction : paramètres, variables, sections de code.

NOTE : Si vous utilisez un DFB qui est déjà dans la bibliothèque définie par l'utilisateur et que vous le modifiez, le nouveau type modifié sera utilisé pour toute instance supplémentaire du projet ouvert. Cependant, la bibliothèque définie par l'utilisateur reste inchangée.

Description du type DFB

Le graphe ci-dessous illustre une représentation graphique d'un modèle DFB.



Le bloc fonction comprend les éléments suivants :

- Nom : nom du type DFB (32 caractères max.) Ce nom doit être unique dans les bibliothèques, les caractères utilisés autorisés dépendent du choix fait dans la zone **Identificateurs** de l'onglet **Extensions de langgage** des **options du projet** *(voir Unity Pro, Modes de marche,)*:
- Entrées : paramètres d'entrée (hors paramètres d'E/S)
- Sorties : paramètres de sortie (hors paramètres d'E/S)
- Entrées/Sorties : paramètres d'E/S.
- Variables publiques : variables internes accessibles par le programme d'application.
- Variables privées : variables internes imbriquées ou DFB, inaccessibles par le programme d'application.
- Sections : sections de code DFB dans LD, IL, ST ou FBD.
- Commentaire (1024 caractères maximum). Les caractères de formatage (retour chariot, onglet, etc.) ne sont pas autorisés.

Pour chaque type de DFB, un fichier descriptif est également accessible par une boîte de dialogue : taille du DFB, nombre de paramètres et variables, numéro de version, date de la dernière modification, niveau de protection, etc.

Aide en ligne pour les types DFB

Il est possible de lier un fichier d'aide HTML à chaque DFB dans la bibliothèque définie par l'utilisateur. Ce fichier doit :

- avoir un nom identique à celui du DFB associé,
- se trouver dans le répertoire *\Schneider Electric\FFBLibset\CustomLib\MyCustomFam* ***Language*** (où ***Language*** correspond à ***Eng***, ***Fre***, ***Ger***, ***Ita***, ***Spa*** ou ***Chs*** selon la langue souhaitée).

Création d'une instance de DFB

Une fois le type de DFB créé, vous pouvez définir une instance de DFB via l'éditeur de variables ou lorsque la fonction est appelée dans l'éditeur de programmes.

Utilisation des instances de DFB

Une instance DFB est utilisée comme suit

- en tant que bloc fonction standard dans un langage à contacts (LD) ou à blocs fonction (FBD),
- en tant que fonction élémentaire dans un langage littéral structuré (ST) ou liste d'instructions (IL).

Une instance DFB peut être utilisée dans toutes les tâches du programme d'application, sauf les tâches événementielles ou les transitions de diagramme fonctionnel en séquence (SFC).

Archivage

Les types DFB créés par l'utilisateur peuvent être stockés *(voir Unity Pro, Modes de marche,)* dans la bibliothèque de fonctions et de blocs fonctions.

Description des Blocs Fonction Utilisateur (DFB)

17

Objet de ce chapitre

Ce chapitre présente les différents éléments qui constituent les blocs fonction utilisateur.

Contenu de ce chapitre

Ce chapitre contient les sujets suivants :

Définition des données internes de blocs fonction dérivés (DFB)

Vue d'ensemble

Il existe deux types de données internes DFB :
- les paramètres : entrées, sorties ou entrées/sorties,
- les variables publiques ou privées.

Les données internes du bloc fonction dérivé sont définies à l'aide de symboles (ces données ne peuvent pas être désignées comme des adresses).

Eléments à définir pour chaque paramètre

Lors de la création du bloc fonction, les éléments suivants doivent être définis pour chaque paramètre :
- Nom : nom du type DFB (32 caractères maximum). Ce nom doit être unique dans les bibliothèques, les caractères utilisés autorisés dépendent du choix fait dans la zone **Identificateurs** de l'onglet **Extensions de langage** des **options du projet** :
- un type d'objet (BOOL, INT, REAL, etc.).
- Commentaire de 1 024 caractères maximum (facultatif). Les caractères de mise en forme (retour chariot, tabulation, etc.) sont interdits),
- une valeur initiale.
- Attribut Lecture/Ecriture : détermine si la variable peut être écrite lors de l'exécution (R : lecture seule - R/W : lecture/écriture). Cet attribut ne doit être défini que pour les variables publiques.
- Attribut de sauvegarde : détermine si la variable peut être enregistrée.

Types d'objets

Les types d'objets disponibles pour les paramètres DFB appartiennent aux familles suivantes :
- Famille de données élémentaires : EDT. Cette famille comprend les types d'objets suivants : Booléen (BOOL, EBOOL), Entier (INT, DINT, etc.), Réel (REAL), Chaîne de caractères (STRING), Chaîne de bits (BYTE, WORD, etc.), etc.
- Famille de données dérivées : DDT. Cette famille comprend les types d'objets tableau (ARRAY) et structure (utilisateur ou IODDT).
- Famille de données génériques : ANY_ARRAY_xxx.
- Famille de bloc fonction : FB. Cette famille comprend les types d'objets EFB et DFB.

Objets autorisés pour les différents paramètres

Pour de meilleures performances, le mode d'adressage des paramètres DFB doit être transféré par adresse pour les familles d'objet suivantes :

- entrées,
- entrées/sorties,
- sorties.

Le mode d'adressage d'un élément d'un bloc fonction est lié au type de l'élément. Les modes d'adressage sont transmis par :

- Valeur (VAL)
- Entrée de table de réaffectation (RTE)
- Adresse logique : RTE+décalage (L-ADR)
- Adresse logique et nombre d'éléments (L-ADR-LG)
- Structure de voie E/S (IOCHS)

Pour chacun des paramètres DFB, les familles d'objets suivantes peuvent être utilisées avec les modes d'adressage associés :

Famille d'objets	**EDT**	**STRING**	**Anonyme ou tableau DDT**	**DDT (1)**	**IODDT**	**GDT : ANY_ARRAY_x**	**FB**	**ANY...**
entrées,	VAL	L-ADR-LG	L-ADR-LG	L-ADR	Non	L-ADR-LG	Non	L-ADR-LG
Entrées/ sorties	L-ADR(2)	L-ADR-LG	L-ADR-LG	L-ADR	IOCHS *(voir page 596)*	L-ADR-LG	Non	L-ADR-LG
Sorties	VAL	VAL	L-ADR-LG	VAL	Non	L-ADR-LG	Non	L-ADR-LG
Variables publiques	VAL	VAL	VAL	VAL	Non	Non	Non	Non
Variables privées	VAL	VAL	VAL	VAL	Non	Non	RTE	Non

Légende :	
(1)	Famille de données dérivées, à l'exception des types de données dérivées d'E/S (IODDT).
(2)	Sauf les variables statiques de type EBOOL, sur les automates Quantum.

⚠ ATTENTION

COMPORTEMENT INATTENDU DE L'APPLICATION - INDEX DE TABLEAU

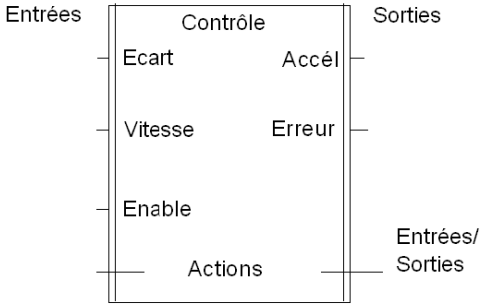
Prenez en compte le décalage de l'index pour les variables ARRAY dotées d'une valeur non nulle pour l'entrée ANY_ARRAY_x de l'index de début (le décalage correspond à la valeur de l'index de début).

Le non-respect de ces instructions peut provoquer des blessures ou des dommages matériels.

Paramètres DFB

Illustration

Cette illustration présente des exemples de paramètres de DFB



Description des paramètres

Ce tableau décrit le rôle de chaque type de paramètres

Paramètre	Nombre maximum	Rôle
Entrées	32 (1)	Ces paramètres permettent de passer des valeurs du programme application vers le programme interne du DFB. Ils sont accessibles en lecture par le DFB, mais ne sont pas accessibles par le programme application.
Sorties	32 (2)	Ces paramètres permettent de passer des valeurs du DFB vers le programme application. Ils sont accessibles en lecture par le programme application sauf pour les paramètres de type ARRAY.
Entrées/ Sorties	32	Ces paramètres permettent de passer des données du programme application vers le DFB qui peut la modifier et la repasser vers le programme application. Ces paramètres ne sont pas accessibles par le programme application.

Légende :

(1) Nombre d'entrées + Nombre d'entrées/sorties inférieur ou égal à 32

(2) Nombre de sorties + Nombre d'entrées/sorties inférieur ou égal à 32

NOTE : Les IODDT associés aux appareils CANopen pour Modicon M340 ne peuvent pas être utilisés comme paramètre d'E/S DFB. Lors de l'étape d'analyse/compilation d'un projet, le message suivant : "This IODDT cannot be used as a DFB parameter" signale les limitations à l'utilisateur.

Paramètres accessibles par le programme application

Les seuls paramètres accessibles par le programme application en dehors de l'appel sont les paramètres de sorties. Pour cela, vous devez utiliser dans le programme la syntaxe suivante : **Nom_DFB.Nom_paramètre**

Nom_DFB représente le nom de l'instance du DFB utilisé (32 caractères au maximum).

Nom_paramètre représente le nom du paramètre de sortie (32 caractères au maximum).

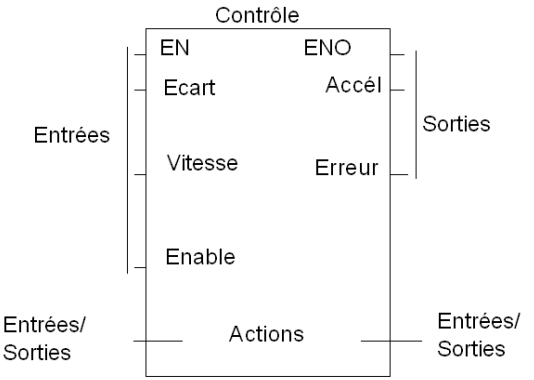
Exemple : `Controle.Accel` indique la sortie `Accel` de l'instance de DFB nommée `Controle`

Paramètres EN et ENO

EN est un paramètre d'entrée et **ENO** un paramètre de sortie. Ils sont de type BOOL et peuvent être ou ne pas être utilisés (facultatif) lors de la définition d'un type de DFB.

Dans le cas où l'utilisateur souhaite les utiliser l'éditeur les positionne automatiquement : EN est le premier paramètre d'entrée et ENO le premier paramètre de sortie.

Exemple d'implémentation des paramètres EN\ENO.



Si le paramètre d'entrée EN d'une instance reçoit la valeur 0 (FALSE) alors :

- la ou les sections constituant le code du DFB ne sont pas exécutées (cela est géré par le système),
- le paramètre de sortie ENO est mis à l'état 0 (FALSE) par le système.

Si le paramètre d'entrée EN d'une instance reçoit la valeur 1 (TRUE) alors :

- la ou les sections constituants le code du DFB sont exécutées (cela est géré par le système),
- le paramètre de sortie ENO est défini sur 1 (TRUE) par le système.

Si une erreur est détectée (erreur process par exemple) par l'instance du DFB, l'utilisateur peut s'il le souhaite définir le paramètre de sortie ENO sur 0 (FALSE). Dans ce cas :

- soit les paramètres de sorties sont figés dans l'état qui était le leur lors du traitement précédant jusqu'à la disparition du défaut,
- soit l'utilisateur prévoit dans le code du DFB un forçage des sorties dans l'état qu'il souhaite jusqu'à la disparition du défaut.

Variable VAR_IN_OUT

Très souvent, on utilise des blocs fonction pour lire une variable au niveau de l'entrée (variables d'entrée), traiter celle-ci, et sortir à nouveau les valeurs modifiées de la même variable (variables de sortie). Ce cas exceptionnel d'une variable d'entrée/de sortie est également appelé variable `VAR_IN_OUT`.

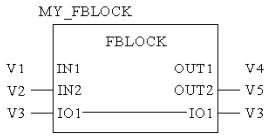
Il convient de noter les particularités suivantes lors de l'utilisation de blocs fonction/DFB avec des variables `VAR_IN_OUT` :

- une variable doit être affectée à toutes les entrées `VAR_IN_OUT`.
- il est interdit d'affecter des valeurs littérales ou des constantes aux entrées `VAR_IN_OUT`.
- **aucune** valeur ne doit être affectée aux sorties `VAR_IN_OUT`.
- les variables `VAR_IN_OUT` ne peuvent **pas** être utilisées en dehors de l'appel du bloc fonction.

Appel d'un bloc fonction avec une variable `VAR_IN_OUT` dans IL :

```
CAL MY_FBLOCK(IN1:=V1, IN2:=V2, IO1:=V3,
              OUT1=>V4, OUT2=>V5)
```

Appel du même bloc fonction dans FBD :



les variables `VAR_IN_OUT` ne peuvent **pas** être utilisées en dehors de l'appel du bloc fonction.

Les appels de blocs fonction suivants sont par conséquent **invalides** :

Appel **invalide**, exemple 1 :

`LD V1`	Chargement des variables `V1` dans l'accumulateur
`CAL InOutFB`	Appel d'un bloc fonction avec un paramètre `VAR_IN_OUT`. L'accumulateur est alors chargé avec référence à un paramètre `VAR_IN_OUT`.
`AND V2`	Liaison ET du contenu de l'accumulateur avec les variables `V2`. **Erreurs :** L'opération ne peut pas être exécutée car il n'est pas possible d'accéder au paramètre `VAR_IN_OUT` (contenu de l'accumulateur) en dehors de l'appel d'un bloc fonction.

Appel **invalide**, exemple 2 :

`LD V1`	Chargement des variables `V1` dans l'accumulateur
`AND InOutFB.inout`	Liaison ET du contenu de l'accumulateur avec référence à un paramètre `VAR_IN_OUT`. **Erreurs :** L'opération ne peut pas être exécutée car il n'est pas possible d'accéder au paramètre `VAR_IN_OUT` en dehors de l'appel d'un bloc fonction.

Les appels de blocs fonction suivants sont en revanche **valides** :

Appel **valide**, exemple 1 :

`CAL InOutFB (IN1:=V1,inout:=V2`	Appel d'un bloc fonction avec un paramètre `VAR_IN_OUT` et affectation des paramètres réels au sein de l'appel de bloc fonction.

Appel **valide**, exemple 2 :

`LD V1`	Chargement des variables `V1` dans l'accumulateur
`ST InOutFB.IN1`	Affectation du contenu de l'accumulateur au paramètre `IN1` du bloc fonction `IN1`.
`CAL InOutFB(inout:=V2)`	Appel du bloc fonction avec affectation du paramètre réel (`V2`) au paramètre `VAR_IN_OUT`.

Variables DFB

Description des variables

Le tableau ci-dessous décrit le rôle de chaque type de variable.

Variable	Nombre maximum	Rôle
Publique	illimité	Ces variables internes du DFB peuvent être utilisées par le DFB, par le programme application et par l'utilisateur en mode réglage.
Privée	illimité	Ces variables internes du DFB peuvent être utilisées uniquement par ce bloc fonction et ne sont par conséquent pas accessibles par le programme application, mais ces types de variables sont accessibles via le tableau d'animation. Ces variables sont généralement des variables nécessaires à la programmation du bloc, mais sans intérêt pour l'utilisateur (résultat d'un calcul intermédiaire, par exemple).

NOTE : les DFB imbriqués sont déclarés comme des variables privées du DFB parent. Ainsi, leurs variables ne sont pas accessibles via la programmation, mais via la table d'animation.

Variables accessibles par le programme application

Les seules variables accessibles par le programme application sont les variables publiques. Pour cela, vous devez utiliser dans le programme la syntaxe suivante : **Nom_DFB.Nom_variable**

Nom_DFB représente le nom de l'instance du DFB utilisé (32 caractères au maximum),

Nom_variable représente le nom de la variable publique (8 caractères au maximum).

Exemple : `Contrôle.Gain` indique la variable publique `Gain` de l'instance de DFB nommée `Contrôle`

Enregistrement des variables publiques

Le réglage sur 1 du bit système `%S94` provoque l'enregistrement des variables publiques que vous avez modifiées par programme ou par réglage, en lieu et place des valeurs initiales de ces variables (définies dans les instances de DFB).

Le remplacement n'est possible que si l'attribut de sauvegarde est correctement défini pour la variable.

ATTENTION

ECHEC DU CHARGEMENT DE L'APPLICATION

Le bit %S94 ne doit pas être réglé sur 1 lors d'un chargement.

Si le bit %S94 est réglé sur 1, le chargement risque d'être impossible.

Le non-respect de ces instructions peut provoquer des dommages matériels.

Section de code DFB

Généralités

La ou les sections de code définissent le traitement que doit effectuer le DFB, en fonction des paramètres déclarés.

Si l'option IEC est positionnée, une seule section peut être attachée au DFB. Dans le cas contraire, un DFB peut contenir plusieurs sections de code; le nombre de sections étant illimité.

Langages de programmation

Pour programmer les sections de DFB vous pouvez utiliser les langages suivants:

- Liste d'instructions IL,
- littéral structuré ST,
- langage à contacts LD,
- Langage à blocs fonctionnels (FBD).

Définition d'une section

Une section est définie par:

- un nom symbolique qui identifie la section (32 caractères au maximum),
- une condition de validation qui définit l'exécution de la section,
- un commentaire (256 caractères au maximum),
- un attribut de protection (pas de protection, section protégée en écriture, section protégée en lecture/écriture).

Règles de programmation

Lorsqu'elle est exécutée, une section de DFB ne peut utiliser que les paramètres que vous avez définis pour le bloc fonction (paramètres d'entrées, de sorties, d'entrées/sorties et variables internes).

Ceci à pour conséquence, qu'un bloc fonction DFB ne peut utiliser les variables globales de l'application, ni les objets d'entrées/sorties, à l'exception des bits et mots système (%Si, %SWi et %SDi).

Une section de DFB a les droits d'accès maximum (lecture et écriture) sur ses paramètres.

Exemple de code

Le programme suivant donne l'exemple code en littéral structuré ST

```
CHR_200:=CHR_100;
CHR_114:=CHR_104;
CHR_116:=CHR_106;
RESET DEMARRE;
(*On incremente 80 fois CHR_100*)
FOR CHR_102:=1 TO 80 DO
          INC CHR_100;
          WHILE((CHR_104-CHR_114)<100)DO
          IF(CHR_104>400) THEN
EXIT;
          END_IF;
          INC CHR_104;
          REPEAT
          IF(CHR_106>300) THEN
EXIT;
          END_IF;
          INC CHR_106;
          UNTIL ((CHR_100-CHR_116)>100)
          END_REPEAT;
          END_WHILE;
          (* On boucle tant que CHR_106)
          IF (CHR_106=CHR_116)
          THEN EXIT;
          ELSE
              CHR_114:=CHR_104;
              CHR_116:=CHR_106;
          END_IF;
          INC CHR_200;
END_FOR;
```

Instances de blocs fonction utilisateur (DFB)

18

Objet de ce chapitre

Ce chapitre présente la création d'une instance DFB et son exécution.

Contenu de ce chapitre

Ce chapitre contient les sujets suivants :

Création d'une instance DFB

Instance de DFB

Une instance de DFB est une copie du modèle de DFB (type de DFB):

- elle exploite le code du type de DFB (le code n'est pas dupliqué).
- elle crée une zone de données spécifique à cette instance, qui est la recopie des paramètres et des variables du type de DFB. Cette zone est située dans l'espace données de l'application.

Vous devez repérer chaque instance de DFB que vous créez, par un nom de 32 caractères au maximum, les caractères utilisés autorisés dépendent du choix fait dans la zone **Identificateurs** de l'onglet **Extensions de langgage** des **options du projet** *(voir Unity Pro, Modes de marche,)*.

Le premier caractère doit être une lettre. Les mots clefs et les symboles sont interdits.

Création d'une instance

A partir d'un type de DFB, vous pouvez créer autant d'instances que nécessaire; la seule limitation étant la taille mémoire de l'automate.

Valeurs initiales

Les valeurs initiales des paramètres et variables publiques, que vous avez définies lors de la création du type de DFB, peuvent être modifiées pour chaque instance du DFB.

Les paramètres des DFB n'ont pas tous une valeur initiale.

Modification des valeurs initiales des éléments dans les instances de DFB

	EDT (sauf type String)	**Type String**	**Structures**	**DDT structure**	**FB**	**ANY_ARRAY**	**IODDT**	**ANY_...**
Entrées	Oui	Non	Non	Non	-	Non	-	Non
Entrées\Sorties	Non	Non	Non	Non	-	Non	Non	Non
Sorties	Oui	Oui	Non	Oui	-	-	-	Non
variables publiques	Oui	Oui	Oui	Oui	-	-	-	-
Variables Privées	Non	Non	Non	Non	Non	-	-	-

Modification des valeurs initiales des éléments dans le type de DFB

	EDT (sauf type String)	**Type String**	**Tableaux**	**DDT structure**	**FB**	**ANY_ARRAY**	**IODDT**	**ANY_...**
Entrées	Oui	Non	Non	Non	-	Non	-	Non
Entrées\Sorties	Non	Non	Non	Non	-	Non	Non	Non
Sorties	Oui	Oui	Non	Oui	-	-	-	Non
variables publiques	Oui	Oui	Oui	Oui	-	-	-	-
Variables Privées	Oui	Oui	Oui	Oui	Non	-	-	-

Exécution d'une instance de DFB

Fonctionnement

Une instance de DFB s'exécute de la manière suivante.

Etape	Action
1	Chargement des valeurs dans les paramètres d'entrées et d'entrées/sorties. Toute entrée sans affectation prend à l'initialisation (ou sur reprise à froid) la valeur initiale définie dans le type de DFB. Elle garde ensuite la dernière valeur qui lui a été affectée.
2	Exécution du programme interne du DFB.
3	Ecriture des paramètres de sorties.

NOTE : Les variables internes des DFB ne sont pas réinitialisées lorsque la commande **Générer le projet en ligne** est exécutée après la modification d'une entrée. Pour réinitialiser toutes les variables internes, utilisez la commande **Regénérer tout le projet**.

Mise au point des DFB

Le logiciel Unity Pro propose plusieurs outils de mise au point des DFB :

- table d'animation : tous les paramètres, variables publiques et privées sont affichés et animés en temps réel (possibilité de modifier et de forcer les objets) ;
- point d'arrêt, pas à pas et diagnostic programme ;
- écrans d'exploitation : pour la mise au point unitaire.

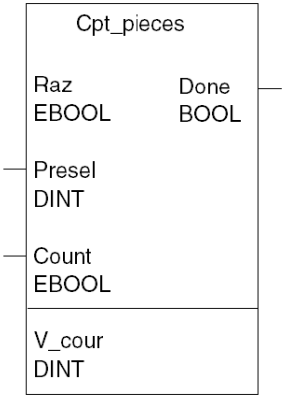
Exemple de programmation d'un bloc DFB

Généralités

Cet exemple de programmation d'un compteur, à partir d'un DFB, est donné à titre didactique.

Caractéristiques du type de DFB

Le type de DFB permettant de réaliser le compteur est le suivant.



Les éléments du Type DFB `Cpt_pieces` sont les suivants.

Eléments	Description
Nom du type de DFB	**Cpt_pièces**
Paramètres d'entrées	• **Raz**: remise à zéro du compteur (type EBOOL) • **Presel** : valeur de présélection du compteur (type DINT) • **Count**: entrée de comptage (type EBOOL)
Paramètres de sorties	**Done**: sortie de valeur de présélection atteinte (type BOOL)
Variable interne publique	**V_cour**: valeur courante du compteur (type DINT)

Fonctionnement du compteur

Le fonctionnement du compteur doit être le suivant.

Phase	Description
1	Le DFB compte les fronts montants sur l'entrée **Count**.
2	Le nombre de fronts comptés est mémorisé par la variable `V_cour`. Cette variable est remise à zéro par un front montant sur l'entrée `Raz`.
3	Lorsque le nombre de fronts comptés est égal à la valeur de présélection, la sortie `Done` est positionnée à 1. Cette variable est remise à zéro par un front montant sur l'entrée `Raz`.

Programme interne du DFB

Le programme interne du type de DFB `Cpt_pieces` est définie en langage ST de la manière suivante.

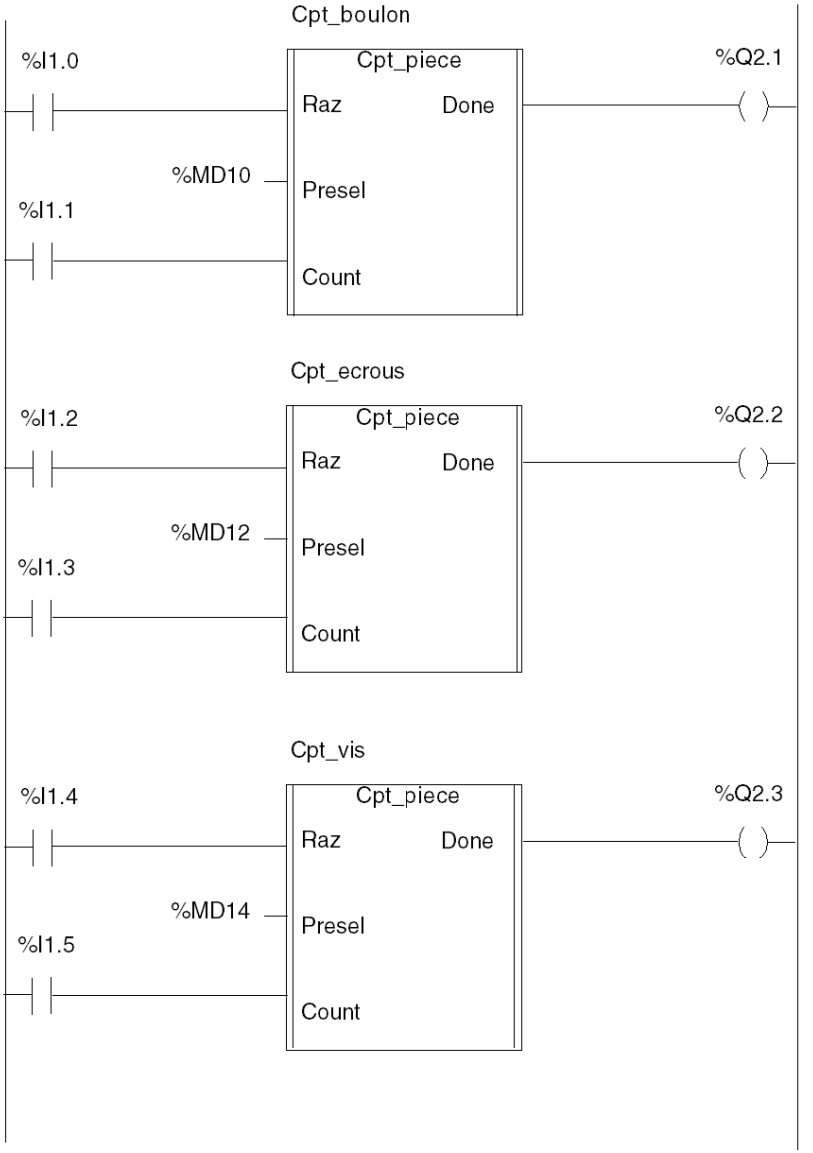
```
!(*Programmation du DFB Cpt_pieces*)
IF RE (Raz) THEN
        V_cour:=0;
END_IF;
IF RE (Count) THEN
        V_cour:=V_cour+1;
END_IF;
IF(V_cour>=Presel) THEN
        SET (Done);
ELSE
        RESET (Done);
END_IF;
```

Exemple d'utilisation

Supposons que votre application nécessite de compter 3 types de pièces (par exemple, des boulons, des écrous et des vis). Vous pouvez utiliser 3 fois le type de DFB `Cpt_pieces` (3 instances) pour réaliser ces différents comptages.

Le nombre de pièces à approvisionner pour chaque type, est défini respectivement dans les mots %MD10, %MD12 et %MD14 Lorsque le nombre de pièces est atteint, le compteur commande une sortie (%Q1.2.1, %Q1.2.2 ou %Q1.2.3) qui pilote l'arrêt du système d'approvisionnement de pièces correspondant.

Le programme application est saisi en langage à contacts de la manière suivante. Il utilise les 3 DFB (instances) `Cpt_boulons`, `Cpt_écrous` et `Cpt_vis` afin de compter les différentes pièces.



Utilisation des DFB à partir de différents langages de programmation

19

Objet de ce chapitre

Ce chapitre présente l'appel des instances de DFB à partir des différents langages de programmation.

Contenu de ce chapitre

Ce chapitre contient les sujets suivants :

Règles d'utilisation des DFB dans un programme

Généralités

Les instances DFB peuvent être utilisées dans tous les langages [liste d'instructions (IL), littéral structuré (ST), langage à contacts (LD) et langage en blocs fonctionnels (FBD)], ainsi que dans toutes les tâches du programme d'application (sections, sous-programmes, etc.), à l'exception des tâches d'événements et des transitions du programme SFC.

Règles générales d'utilisation

Lorsque vous utilisez un DFB, vous devez respecter les consignes suivantes, quel que soit le langage utilisé :

- Il n'est pas nécessaire de définir tous les paramètres d'entrée, de sortie ou d'entrée/sortie, sauf les paramètres obligatoires suivants :
  - les paramètres d'entrées de type données génériques (ANY_INT, ANY_ARRAY,...),
  - des paramètres d'entrée/de sortie,
  - les paramètres de sorties de type données (hors tableaux) génériques (ANY_INT, ANY_REAL,...).
  - paramètres d'entrée d'éléments de type STRING,
- les paramètres d'entrées non câblés gardent la valeur du précédent appel ou la valeur d'initialisation définies pour ces paramètres, si le bloc n'a jamais été appelé,
- Tous les objets affectés aux paramètres d'entrée, de sortie et d'entrée/sortie doivent être du même type que celui des objets qui ont été définis à la création du type DFB (par exemple : si le type INT est défini pour le paramètre d'entrée "vitesse", vous ne pouvez pas y affecter le type DINT ou REAL).
  Les seules exceptions concernent les types BOOL et EBOOL pour les paramètres d'entrée et de sortie (pas les paramètres d'entrée/sortie), qui sont utilisés en même temps.
  Exemple : Le paramètre d'entrée "Validation" peut être défini comme BOOL et associé à un bit interne %Mi de type EBOOL. Toutefois, le paramètre d'entrée possède des propriétés de type BOOL dans le code interne du type DFB (ne gère pas les fronts).

Affectation des paramètres

Le tableau suivant résume les différentes possibilités d'affectation des paramètres possibles dans les différents langages de programmation.

Paramètre	Type	Affectation du paramètre (1)	Affectation
Entrées	EDT (2)	Défini, valeur, objet ou expression	Facultatif (3)
	BOOL	Défini, valeur, objet ou expression	Optionnelle
	DDT	Défini, valeur ou objet	Obligatoire
	ANY_...	Défini ou objet	Obligatoire
	ANY_ARRAY	Défini ou objet	Obligatoire
Entrées/sorties	EDT	Défini ou objet	Obligatoire
	DDT	Défini ou objet	Obligatoire
	IODDT	Défini ou objet	Obligatoire
	ANY_...	Défini ou objet	Obligatoire
	ANY_ARRAY	Défini ou objet	Obligatoire
Sorties	EDT	Défini ou objet	Optionnelle
	DDT	Défini ou objet	Optionnelle
	ANY_...	Défini ou objet	Obligatoire
	ANY_ARRAY	Défini ou objet	Optionnelle

(1) Défini en langage à contacts (LD) ou langage en blocs fonctionnels (FBD). Valeur ou objet en langage littéral structuré (ST) ou liste d'instructions (IL).

(2) Sauf paramètres de type BOOL

(3) Sauf paramètres de type STRING qui sont obligatoires

Affectation des paramètres

Le tableau suivant résume les différentes possibilités d'affectation des paramètres possibles dans les différents langages de programmation.

Paramètre	Type	Affectation du paramètre (1)	Affectation
Entrées	EDT (2)	Défini, valeur, objet ou expression	Facultatif (3)
	BOOL	Défini, valeur, objet ou expression	Optionnelle
	DDT	Défini, valeur ou objet	Obligatoire
	ANY_...	Défini ou objet	Obligatoire
	ANY_ARRAY	Défini ou objet	Obligatoire
Entrées/sorties	EDT	Défini ou objet	Obligatoire
	DDT	Défini ou objet	Obligatoire
	IODDT	Défini ou objet	Obligatoire
	ANY_...	Défini ou objet	Obligatoire
	ANY_ARRAY	Défini ou objet	Obligatoire
Sorties	EDT	Défini ou objet	Optionnelle
	DDT	Défini ou objet	Optionnelle
	ANY_...	Défini ou objet	Obligatoire
	ANY_ARRAY	Défini ou objet	Optionnelle

(1) Défini en langage à contacts (LD) ou langage en blocs fonctionnels (FBD). Valeur ou objet en langage littéral structuré (ST) ou liste d'instructions (IL).

(2) Sauf paramètres de type BOOL

(3) Sauf paramètres de type STRING qui sont obligatoires

Règles d'utilisation des tableaux avec des DFB

⚠ AVERTISSEMENT

FONCTIONNEMENT INATTENDU DE L'EQUIPEMENT

Vérifiez la dimension des tableaux lors de la copie de tableau source vers cible en DBF.

Le non-respect de ces instructions peut provoquer la mort, des blessures graves ou des dommages matériels.

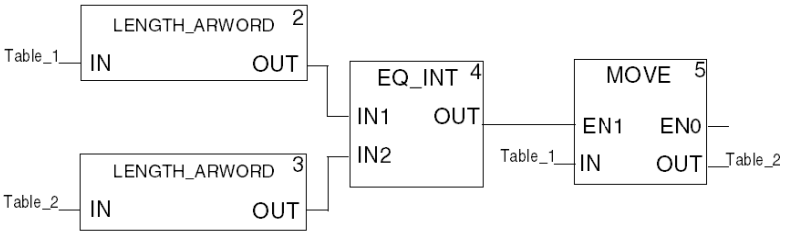
Pour utiliser des tableaux dynamiques, il est obligatoire de vérifier que les dimensions des tableaux sont identiques. Dans un cas particulier, lors de l'utilisation de tableaux dynamiques en sortie ou entrée/sortie, un débordement peut conduire à une exécution incorrecte du programme qui bloquerait l'automate.

Ce comportement apparaît si les conditions ci-dessous sont simultanément remplies :

- Utilisation d'un DBF avec au moins un paramètre de sortie ou d'E/S de type tableau dynamique (ANY_ARRAY_XXX).
- Dans le codage d'un DBF, utilisez une fonction ou un bloc de fonctions (FFB de type FIFO, LIFO, MOVE, MVX, T2T, SAH ou SEL). Remarquez que la fonction ou le FFB nécessitent deux paramètres de type ANY dont au moins un est défini dans la sortie.
- Le paramètre DBF du tableau dynamique est utilisé pour l'écriture lors de l'appel FFB (sur le paramètre de type ANY). Pour d'autres paramètres de type ANY, c'est un tableau fixe qui est utilisé.
- La dimension du tableau de dimension fixe est supérieure à la dimension du tableau dynamique calculé pour enregistrer le résultat.

Exemple de vérification de dimension des tableaux

L'exemple ci-dessous montre comment vérifier la dimension des tableaux à l'aide de la fonction LENGTH_ARWORD dans un DBF.



Dans cet exemple, Table_1 est un tableau de dimension fixe, Table_2 est un tableau dynamique de type ANY_ARRAY_WORD. Ce programme vérifier la dimension de chaque tableau. Les fonctions LENGTH_ARWORD calculent la dimension de chaque tableau pour conditionner l'exécution de la fonction MOVE.

Utilisation des IODDT dans un DFB

Présentation

Les tableaux suivants présentent les différents IODDT pour les automates Modicon M340, Premium et Quantum qui peuvent être utilisés dans un DFB (exclusivement en tant que paramètres d'entrée/sortie *(voir page 573)*).

IOODT utilisables dans un DFB

Le tableau suivant présente les IODDT des différentes applications pour automates Modicon M340, Premium et Quantum qui peuvent être utilisés dans un DFB.

Familles d'IODDT	Modicon M340	Premium	Quantum
Application TOR			
T_DIS_IN_GEN	Aucune	Aucune	Aucune
T_DIS_IN_STD	Aucune	Aucune	Aucune
T_DIS_EVT	Aucune	Aucune	Aucune
T_DIS_OUT_GEN	Aucune	Aucune	Aucune
T_DIS_OUT_STD	Aucune	Aucune	Aucune
T_DIS_OUT_REFLEX	Aucune	Aucune	Aucune
Application analogique			
T_ANA_IN_GEN	Aucune	Aucune	Aucune
T_ANA_IN_STD	Aucune	Aucune	Aucune
T_ANA_IN_CTRL	Aucune	Oui	Aucune
T_ANA_IN_EVT	Aucune	Oui	Aucune
T_ANA_OUT_GEN	Aucune	Aucune	Aucune
T_ANA_OUT_STD	Aucune	Aucune	Aucune
T_ANA_IN_BMX	Oui	Aucune	Aucune
T_ANA_IN_T_BMX	Oui	Aucune	Aucune
T_ANA_OUT_BMX	Oui	Aucune	Aucune
T_ANA_IN_VE	Aucune	Aucune	Aucune
T_ANA_IN_VWE	Aucune	Aucune	Aucune
T_ANA_BI_VWE	Aucune	Aucune	Aucune
T_ANA_BI_IN_VWE	Aucune	Aucune	Aucune
Application de comptage			
T_COUNT_ACQ	Aucune	Oui	Aucune
T_COUNT_HIGH_SPEED	Aucune	Oui	Aucune
T_COUNT_STD	Aucune	Oui	Aucune

Familles d'IODDT	Modicon M340	Premium	Quantum
T_SIGN_CPT_BMX	Oui	Aucune	Aucune
T_UNSIGN_CPT_BMX	Oui	Aucune	Aucune
T_CNT_105	Aucune	Aucune	Aucune
Application de came électronique			
T_CCY_GROUP0	Aucune	Aucune	Aucune
T_CCY_GROUP1_2_3	Aucune	Aucune	Aucune
Application de commande d'axes			
T_AXIS_AUTO	Aucune	Oui	Aucune
T_AXIS_STD	Aucune	Oui	Aucune
T_INTERPO_STD	Aucune	Oui	Aucune
T_STEPPER_STD	Aucune	Oui	Aucune
Application Sercos			
T_CSY_CMD	Aucune	Oui	Aucune
T_CSY_TRF	Aucune	Oui	Aucune
T_CSY_RING	Aucune	Oui	Aucune
T_CSY_IND	Aucune	Oui	Aucune
T_CSY_FOLLOW	Aucune	Oui	Aucune
T_CSY_COORD	Aucune	Oui	Aucune
T_CSY_CAM	Aucune	Oui	Aucune
Application de communication			
T_COM_STS_GEN	Oui	Oui	Aucune
T_COM_UTW_M	Aucune	Oui	Aucune
T_COM_UTW_S	Aucune	Oui	Aucune
T_COM_MB	Aucune	Oui	Aucune
T_COM_CHAR	Aucune	Oui	Aucune
T_COM_FPW	Aucune	Oui	Aucune
T_COM_MBP	Aucune	Oui	Aucune
T_COM_JNET	Aucune	Oui	Aucune
T_COM_ASI	Aucune	Oui	Aucune
T_COM_ETY_1X0	Aucune	Oui	Aucune
T_COM_ETY_210	Aucune	Oui	Aucune
T_COM_IBS_128	Aucune	Oui	Aucune
T_COM_IBS_242	Aucune	Oui	Aucune
T_COM_PBY	Aucune	Oui	Aucune

Familles d'IODDT	Modicon M340	Premium	Quantum
T_COM_CPP100	Aucune	Oui	Aucune
T_COM_ETYX103	Aucune	Oui	Aucune
T_COM_ETHCOPRO	Aucune	Oui	Aucune
T_COM_MB_BMX	Oui	Aucune	Aucune
T_COM_CHAR_BMX	Oui	Aucune	Aucune
T_COM_CO_BMX	Oui	Aucune	Aucune
T_COM_ETH_BMX	Oui	Aucune	Aucune
Application de régulation			
T_PROC_PLOOP	Aucune	Oui	Aucune
T_PROC_3SING_LOOP	Aucune	Oui	Aucune
T_PROC_CASC_LOOP	Aucune	Oui	Aucune
T_PROC_SPP	Aucune	Oui	Aucune
T_PROC_CONST_LOOP	Aucune	Oui	Aucune
Application de pesage			
T_WEIGHING_ISPY101	Aucune	Oui	Aucune
Commun aux applications			
T_GEN_MOD	Aucune	Aucune	Aucune

Utilisation d'un bloc DFB dans un programme en langage à contacts

Principe

En langage à contacts LD, il y a deux possibilités pour appeler un bloc fonction DFB:

- par l'intermédiaire d'un appel textuel dans un bloc opération dans lequel la syntaxe et les contraintes sur les paramètres sont identiques à celles du langage littéral structuré ST ;
- par un appel graphique

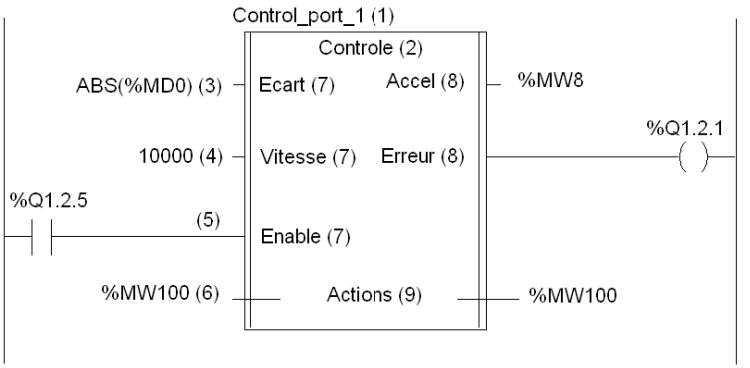
Les entrées des blocs fonction peuvent être câblées ou affectées d'une valeur, d'un objet ou d'une expression. Dans tous les cas, le type de l'élément extérieur (valeur, évaluation de l'expression,...) doit être identique à celui du paramètre d'entrée.

Un bloc DFB doit avoir au moins une entrée booléenne câblée et une sortie (si nécessaire). Pour cela, vous pouvez utiliser les paramètres d'entrée EN et le paramètre de sortie ENO (voir la description des paramètres ci-après).

Vous devez obligatoirement câbler ou affecter les entrées de type ANY_ARRAY, les sorties de type données génériques (ANY_...) et les entrées/sorties d'un bloc DFB.

Représentation graphique d'un bloc DFB

L'illustration suivante présente un exemple simple de programmation d'un DFB.



Eléments du bloc DFB

Le tableau ci-après liste les différents éléments du bloc DFB, repérés dans l'illustration précédente.

Etiquette	Elément
1	Nom du DFB (instance)
2	Nom du type de DFB
3	Entrée affectée par une expression
4	Entrée affectée par une valeur
5	Entrée câblée
6	Entrée affectée par un objet (adresse ou symbole)
7	Paramètres d'entrées
8	Paramètres de sorties
9	Paramètres d'entrées/sorties

Utilisation des paramètres EN\ENO

Voir *Paramètres EN et ENO, page 575*

Utilisation d'un bloc DFB dans un programme en langage littéral structuré

Principe

En littéral structuré ST, l'appel d'un bloc fonction utilisateur s'effectue par un appel du DFB: nom de l'instance de DFB suivi d'une liste d'arguments. A l'intérieur de la liste, matérialisée par des parenthèses, les arguments sont séparés par des virgules.

L'appel du DFB peut être de deux types :

- appel formel, lorsque les arguments sont des affectations (paramètre = valeur). Dans ce cas, l'ordre de saisie des arguments dans la liste est sans importance. Vous pouvez utiliser le paramètre d'entrée EN et le paramètre de sortie ENO pour commander l'exécution du bloc fonction,
- appel informel, lorsque les arguments sont des valeurs (expression, objet ou une valeur immédiate). Dans ce cas, l'ordre de saisie des arguments dans la liste doit respecter l'ordre des paramètres d'entrées du DFB, y compris pour celles qui sont non affectées (l'argument est un champ vide).
  L'utilisation des paramètres EN et ENO n'est pas possible.

```
Nom_DFB (argument 1,argument 2,....,argument n)
```

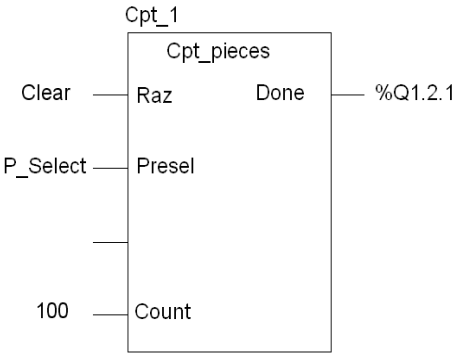
NOTE : Les entrées de type ANY_ARRAY, les sorties de type données génériques (ANY_...) et les entrées/sorties d'un DFB doivent être obligatoirement affectées.

Utilisation des paramètres EN\ENO

Voir *Paramètres EN et ENO, page 575*

Exemple de DFB

L'exemple simple suivant va permettre de comprendre les différents appels d'un DFB en langage littéral structuré. Il s'agit de l'instance `Cpt_1` de `Cpt_parts` : type de DFB.



Appel formel du DFB

L'appel formel du DFB `Cpt_1` s'effectue suivant la syntaxe :

```
Cpt_1 (Raz:=Clear, Presel:=P_Select, Count:=100,
Done=>%Q1.2.1);
```

Cas où seuls les paramètres d'entrées affectés par une valeur (expression, objet ou valeur immédiate) sont saisis dans la liste des arguments.

```
Cpt_1 (Raz:=Clear, Presel:=P_Select, Count:=100);
...
%Q1.2.1:=Cpt_1.Done;
```

Eléments de la phrase

Le tableau suivant liste les différents éléments de la phrase de programme, lors d'un appel formel du DFB.

Elément	Signification
Cpt_1	Instance de nom du DFB
Raz, Presel, Count	Paramètres d'entrées
:=	Symbole d'affectation d'une entrée
Clear	Objet d'affectation d'une entrée (symbole)
100	Valeur d'affectation d'une entrée
Done	Paramètre de sortie
=>	Symbole d'affectation d'une sortie
%Q1.2.1	Objet d'affectation d'une sortie (adresse)
;	Symbole de fin de phrase
,	Symbole de séparation des arguments

Appel informel du DFB

L'appel informel du DFB `Cpt_1` s'effectue suivant la syntaxe :

```
Cpt_1 (Clear, %MD10, , 100);
...
%Q1.2.1:=Cpt_1.Done;
```

Eléments de la phrase

Le tableau suivant liste les différents éléments de la phrase de programme, lors d'un appel formel du DFB.

Elément	Signification
Cpt_1	Instance de nom du DFB
Clear, %MD10, ,100	Objet ou valeur d'affectation des entrées. Les entrées non affectées sont représentées par un champ vide
;	Symbole de fin de phrase
,	Symbole de séparation des arguments

Utilisation d'un DFB dans un programme en liste d'instructions

Principe

En liste d'instructions IL, l'appel d'un bloc fonction utilisateur s'effectue par une instruction `CAL` suivie du nom de l'instance de DFB comme opérande et d'une liste d'arguments (optionnelle). A l'intérieur de la liste, matérialisée par des parenthèses, les arguments sont séparés par des virgules.

Il y a 3 possibilités pour appeler un DFB en langage IL:

- l'instruction `CAL nom_DFB` est suivie d'une liste d'arguments qui sont des affectations (paramètre = valeur). Dans ce cas, l'ordre de saisie des arguments dans la liste est sans importance.
  Vous pouvez utiliser l'entrée EN pour commander l'exécution du bloc fonction.
- L'instruction `CAL nom_DFB` est suivie d'une liste d'arguments qui sont des valeurs (expression, objet ou valeur immédiate). Dans ce cas, l'ordre de saisi des arguments dans la liste doit respecter l'ordre des paramètres d'entrées du DFB, y compris pour celles qui sont non affectées (l'argument est un champ vide). L'utilisation des paramètres EN et ENO n'est pas possible.
- L'instruction CAL nom_DFB n'est pas suivie par une liste d'arguments. Dans ce cas, cette instruction doit être précédée de l'affectation des paramètres d'entrées au travers d'un registre: chargement de la valeur (Load) puis affectation au paramètre d'entrée (Store). L'ordre d'affectation des paramètres (LD/ST) n'a pas d'importance; toutefois, vous devez affecter tous les paramètres d'entrées qui le nécessitent avant d'exécuter la commande `CAL`. L'utilisation des paramètres EN et ENO n'est pas possible.

```
CAL Nom_DFB (argument 1, argument 2, ..., argument n)
```

ou

```
LD Valeur 1
ST Paramètre 1
...
LD Valeur n
ST Paramètre n
CAL Nom_DFB
```

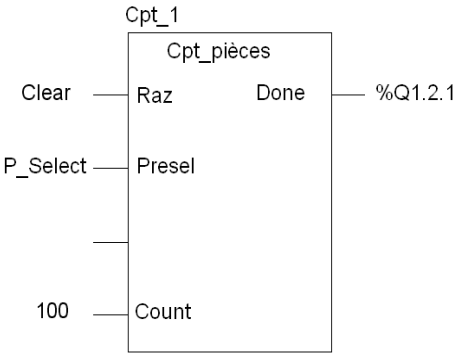
NOTE : Les entrées de type ANY_ARRAY, les sorties de type données génériques (ANY_...) et les entrées/sorties d'un DFB doivent être obligatoirement affectées.

Utilisation des paramètres EN\ENO

Voir *Paramètres EN et ENO, page 575*.

Exemple de DFB

L'exemple suivant va permettre de comprendre les différents appels d'un DFB en langage liste d'instructions. Il s'agit de l'instance `Cpt_1` de `Cpt_parts` : type de DFB



Appel du DFB lorsque les arguments sont des affectations

Lorsque les arguments sont des affectations, l'appel du DFB `Cpt_1` s'effectue suivant la syntaxe :

```
CAL Cpt_1 (Raz:=Clear, Presel:=%MD10, Count:=100,
Done=>%Q1.2.1)
```

Cas où seuls les paramètres d'entrées affectés par une valeur (expression, objet ou valeur immédiate) sont saisis dans la liste des arguments:

```
CAL Cpt_1 (Raz:=Clear, Presel:=%MD10, Count:=100)
...
LD Cpt_1.Done
ST %Q1.2.1
```

Afin de rendre plus lisible votre programme application, vous pouvez saisir un retour à la ligne après les virgules de séparation des arguments. La phrase prend alors la syntaxe suivante:

```
CAL Cpt_1
Raz:=Clear,
Presel:=%MD10,
Count:=100,
Done=>%Q1.2.1)
```

Eléments du programme d'appel du DFB

Le tableau suivant liste les différents éléments du programme d'appel du DFB.

Elément	Signification
CAL	Instruction d'appel du DFB
Cpt_1	Instance de nom du DFB
Raz, Presel, Count	Paramètres d'entrées
:=	Symbole d'affectation d'une entrée
Clear, %MD10, 100	Objet ou valeur d'affectation des entrées
Done	Paramètre de sortie
=>	Symbole d'affectation d'une sortie
%Q1.2.1	Objet d'affectation d'une sortie
,	Symbole de séparation des arguments

Appel du DFB lorsque les arguments sont des valeurs

Lorsque les arguments sont des valeurs, l'appel du DFB `Cpt_1` s'effectue suivant la syntaxe :

```
CAL Cpt_1 (Clear, %MD10,, 100)
...
LD Cpt_1.Done
ST %Q1.2.1
```

Eléments du programme d'appel du DFB

Le tableau suivant liste les différents éléments du programme d'appel du DFB.

Elément	Signification
CAL	Instruction d'appel du DFB
Cpt_1	Instance de nom du DFB
Clear, %MD10, 100	Objet ou valeur d'affectation des entrées
,	Symbole de séparation des arguments

Appel d'un DFB sans argument

Lorsqu'il n'y a pas d'argument, l'appel du DFB `Cpt_1` s'effectue suivant la syntaxe :

```
LD Clear
ST Cpt_1.Raz
LD %MD10
ST Cpt_1.Presel
LD 100
ST Cpt_1.Count
CAL Cpt_1
...
LD Cpt_1.Done
ST %Q1.2.1
```

Eléments du programme d'appel du DFB

Le tableau suivant liste les différents éléments du programme d'appel du DFB.

Elément	Signification
LD Clear	Instruction de chargement de la valeur `Clear` dans un registre
ST Cpt_1.Raz	Instruction d'affectation du contenu du registre au paramètre d'entrée `Cpt_1.Raz`
CAL Cpt_1	Instruction d'appel du DFB `Cpt_1`

Utilisation d'un bloc DFB dans un programme en langage à blocs fonction

Principe

En langage FBD (Diagramme de Blocs Fonction), les blocs fonction utilisateur sont représentés de la même manière qu'en langage à contacts et appelés de manière graphique.

Les entrées des blocs fonction utilisateur peuvent être câblées ou affectées d'une valeur immédiate, d'un objet ou d'une expression. Dans tous les cas, le type de l'élément extérieur doit être identique à celui du paramètre d'entrée.

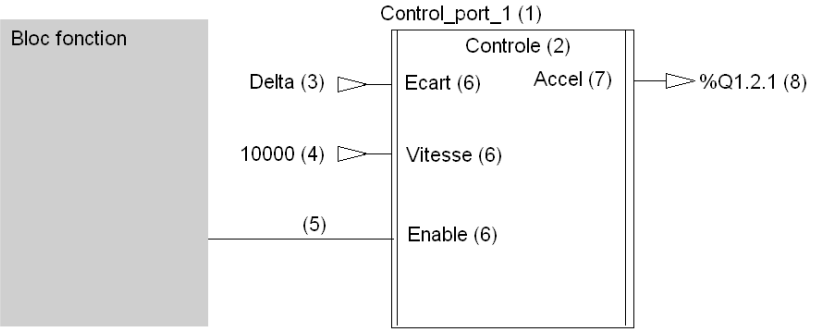
Vous ne pouvez affecter qu'un seul objet (lien vers un autre bloc au variable) sur une entrée du DFB. Par contre, plusieurs objets peuvent être connectés à une même sortie.

Un bloc DFB doit avoir au moins une entrée booléenne câblée et une sortie (si nécessaire). Pour cela, vous pouvez utiliser un paramètre d'entrée EN et un paramètre de sortie ENO.

Vous devez obligatoirement câbler ou affecter les entrées de type ANY_ARRAY, les sorties de type données génériques (ANY_...) et les entrées/sorties d'un bloc DFB.

Représentation graphique d'un bloc DFB

L'illustration suivante présente un exemple simple de programmation d'un DFB.



Eléments du bloc DFB

Le tableau ci-après liste les différents éléments du bloc DFB, repérés dans l'illustration précédente.

Etiquette	Elément
1	Nom du DFB (instance)
2	Nom du type de DFB
3	Entrée affectée par un objet (symbole)
4	Entrée affectée par une valeur
5	Entrée câblée
6	Paramètres d'entrées
7	Paramètre de sortie
8	Sortie affectée par un objet (adresse)

Utilisation des paramètres EN\ENO

Voir *Paramètres EN et ENO, page 575*.

DFB de fiagnostic utilisateur

20

Présentation des DFB de diagnostic utilisateur

Général

Le logiciel Unity Pro vous permet de créer vos propres DFB de diagnostic *(voir Unity Pro, Modes de marche,)*.

Ces DFB de diagnostic sont des DFB standard que vous aurez au préalable configurés avec la **propriété Diagnostic** et dans lesquels vous aurez utilisé les deux fonctions suivantes :

- REGDFB *(voir Unity Pro, Diagnostic, Bibliothèque de blocs)* pour l'enregistrement de la datation d'alarme
- DEREG *(voir Unity Pro, Diagnostic, Bibliothèque de blocs)* pour annuler l'enregistrement de l'alarme

NOTE : Il est fortement recommandé de ne programmer une instance DFB de diagnostic que lorsque vous êtes dans l'application.

Ces DFB vous permettent de surveiller le processus. Ils vont consigner automatiquement les informations choisies dans le Viewer. Vous pouvez ainsi surveiller des changements d'état ou des variations dans le processus.

Avantages

Les principaux avantages de ce service sont les suivants :

- Le diagnostic est intégré au projet, il peut ainsi être pensé au moment du développement et ainsi répondre au mieux aux besoins de l'exploitant.
- Le système de datation et d'enregistrement des erreurs s'effectue à la source (dans l'automate), ainsi les informations représentent exactement l'état du process.
- Vous pouvez connecter plusieurs Viewers (Unity Pro, Magelis, Factory Cast) qui transcriront à l'exploitant l'état exact du process. Chaque Viewer est indépendant et toute action effectuée sur l'un (un acquittement, par exemple) est automatiquement visualisé sur les autres.

Annexes

Objet de ce chapitre

L'annexe contient des informations complémentaires.

Contenu de cette annexe

Cette annexe contient les chapitres suivants :

Codes d'erreur et valeurs EFB

A

Introduction

Les tableaux présentés dans cette section répertorient les codes et les valeurs d'erreur générés pour les EFB, par bibliothèque et par famille.

Contenu de ce chapitre

Ce chapitre contient les sujets suivants :

Tableaux des codes d'erreur pour la bibliothèque de base

Introduction

Les tableaux ci-dessous répertorient les codes et les valeurs d'erreur générés pour les EFB de la bibliothèque de base.

Date & Durée

Tableau des codes et valeurs d'erreur générés pour les EFB de la famille `Date & Durée`.

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
DIVTIME	E_DIVIDE_BY_ZERO	F	-30 176	16#8A20	Division par zéro
DIVTIME	E_NEGATIVE_INPUT_ FOR_TIME_OPERATION	F	-30177	16#8A1F	Une valeur négative ne peut être convertie en données de type `Durée`
DIVTIME	E_ARITHMETIC_ ERROR	F	-30 170	16#8A26	Erreur arithmétique
DIVTIME	E_ERR_ARITHMETIC	F	-30 003	16#8ACD	Dépassement arithmétique (%S18 activé)
DIVTIME	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
MULTIME	E_ERR_ARITHMETIC	F	-30 003	16#8ACD	Dépassement arithmétique (%S18 activé)
MULTIME	E_ARITHMETIC_ERROR_ MUL_OV	F	-30 172	16#8A24	Erreur arithmétique / Dépassement multiplication
MULTIME	E_ARITHMETIC_ERROR_ ADD_OV	F	-30 173	16#8A23	Erreur arithmétique / Dépassement addition
MULTIME	E_ARITHMETIC_ERROR_ BIG_PAR	F	-30 171	16#8A25	Erreur arithmétique / Paramètre dépassant la plage
MULTIME	E_NEGATIVE_INPUT_FOR _TIME_OPERATION	F	-30 177	16#8A1F	Une valeur négative ne peut être convertie en données de type `Durée`
MULTIME	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*

Statistiques

Tableau des codes et valeurs d'erreur générés pour les EFB de la famille `Statistiques`.

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
AVE	E_INPUT_VALUE_OUT_OF_RANGE	F	-30 183	16#8A19	Valeur d'entrée hors limites
AVE	E_DIVIDE_BY_ZERO	F	-30 176	16#8A20	Division par zéro
AVE	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
AVE	E_ARITHMETIC_ ERROR	F	-30 170	16#8A26	Erreur arithmétique
AVE	E_FP_STATUS_FAILED	F	-30 150	16#8A3A	Opération incorrecte de virgule flottante
AVE	E_ARITHMETIC_ ERROR_MUL_OV	F	-30 172	16#8A24	Erreur arithmétique / Dépassement multiplication
AVE	E_ARITHMETIC_ ERROR_ADD_OV	F	-30 173	16#8A23	Erreur arithmétique / Dépassement addition
AVE	E_ARITHMETIC_ ERROR_BIG_PAR	F	-30 171	16#8A25	Erreur arithmétique / Paramètre dépassant la plage
AVE	E_ARITHMETIC_ERROR_UNSIGN_OV	F	-30 174	16#8A22	Erreur arithmétique / Dépassement non signé
MAX	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
MIN	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
MUX	E_SELECTOR_OUT_OF_RANGE	F	-30 175	16#8A21	Sélecteur hors limites

Tableaux des codes d'erreur pour la bibliothèque de diagnostic

Introduction

Les tableaux ci-dessous répertorient les codes et les valeurs d'erreur générés pour les EFB de la bibliothèque de diagnostic.

Diagnostic

Tableau des codes et valeurs d'erreur générés pour les EFB de la famille `Diagnostic`.

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
ONLEVT	E_EFB_ONLEVT	V/F	-30 196	16#8A0C	Erreur d'EFB ONLEVT Etats ENO • Vrai = enregistrement des erreurs OK • Faux = échec de l'enregistrement des erreurs

Tableau des codes d'erreur pour la bibliothèque de communications

Introduction

Le tableau suivant répertorie les codes et les valeurs d'erreur générés pour les EFB de la bibliothèque de communications.

Booléen étendu

Tableau des codes et valeurs d'erreur générés pour les EFB du type `Booléen étendu`.

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
CREAD_REG	E_EFB_MSTR_ERROR	F	-30 191	16#8A11	Erreur de communication MSTR
CREAD_REG	E_EFB_NOT_STATE_RAM_4X	F	-30 531	16#88BD	Variable non affectée à la zone %MW (4x)
CREAD_REG	-	F	8 195	16#2003	Valeur affichée dans le mot d'état (apparaît avec E_EFB_MSTR_ERROR)
CREAD_REG	-	F	8 206	16#200E	Valeur affichée dans le mot d'état (apparaît avec E_EFB_NOT_STATE_RAM_4X)

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
CREAD_REG	-	F	-	-	Voir les tableaux suivants : ● Codes d'erreur Modbus Plus et EtherNet SY/MAX *(voir Modicon Quantum avec Unity, Modules réseau Ethernet, Manuel utilisateur)* ● Codes d'erreur spécifiques SY/MAX *(voir Modicon Quantum avec Unity, Modules réseau Ethernet, Manuel utilisateur)* ● Codes d'erreur EtherNet TCP/IP *(voir Modicon Quantum avec Unity, Modules réseau Ethernet, Manuel utilisateur)*
CWRITE_REG	E_EFB_MSTR_ERROR	F	-30 191	16#8A11	Erreur de communication MSTR
CWRITE_REG	-	F	8 195	16#2003	Valeur affichée dans le mot d'état (apparaît avec E_EFB_MSTR_ERROR)
CWRITE_REG	-	F	8 206	16#200E	Valeur affichée dans le mot d'état (apparaît avec E_EFB_NOT_STATE_RAM_4X)

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
CWRITE_REG	-	F	-	-	Voir les tableaux suivants : ● Codes d'erreur Modbus Plus et EtherNet SY/MAX *(voir Modicon Quantum avec Unity, Modules réseau Ethernet, Manuel utilisateur)* ● Codes d'erreur spécifiques SY/MAX *(voir Modicon Quantum avec Unity, Modules réseau Ethernet, Manuel utilisateur)* ● Codes d'erreur EtherNet TCP/IP *(voir Modicon Quantum avec Unity, Modules réseau Ethernet, Manuel utilisateur)*
MBP_MSTR	E_EFB_OUT_OF_RANGE	F	-30 192	16#8A10	Erreur interne : L'EFB a détecté une violation, par exemple l'écriture a dépassé les limites %MW (4x)
MBP_MSTR	E_EFB_NOT_STATE_RAM_4X	F	-30 531	16#88BD	Variable non affectée à la zone %MW (4x)
MBP_MSTR	-	F	8 195	16#2003	Valeur affichée dans le mot d'état (apparaît avec E_EFB_MSTR_ERROR dans l'état du bloc de commande)
MBP_MSTR	-	F	8 206	16#200E	Valeur affichée dans le mot d'état (apparaît avec E_EFB_NOT_STATE_RAM_4X dans l'état du bloc de commande)

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
MBP_MSTR	-	F	-	-	Voir les tableaux suivants : ● Codes d'erreur Modbus Plus et EtherNet SY/MAX *(voir Modicon Quantum avec Unity, Modules réseau Ethernet, Manuel utilisateur)* ● Codes d'erreur spécifiques SY/MAX *(voir Modicon Quantum avec Unity, Modules réseau Ethernet, Manuel utilisateur)* ● Codes d'erreur EtherNet TCP/IP *(voir Modicon Quantum avec Unity, Modules réseau Ethernet, Manuel utilisateur)*
READ_REG	W_WARN_OUT_OF_RANGE	F	30 110	16#759E	Paramètre hors limites
READ_REG	E_EFB_NOT_STATE_RAM_4X	F	-30 531	16#88BD	Variable non affectée à la zone %MW (4x)
READ_REG	E_EFB_MSTR_ERROR	F	-30 191	16#8A11	Erreur de communication MSTR
READ_REG	-	F	8 195	16#2003	Valeur affichée dans le mot d'état (apparaît avec W_WARN_OUT_OF_RANGE)
READ_REG	MBPUNLOC	F	8 206	16#200E	Valeur affichée dans le mot d'état (apparaît avec E_EFB_NOT_STATE_RAM_4X)

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
READ_REG	-	F	-	-	Voir les tableaux suivants : ● Codes d'erreur Modbus Plus et EtherNet SY/MAX *(voir Modicon Quantum avec Unity, Modules réseau Ethernet, Manuel utilisateur)* ● Codes d'erreur spécifiques SY/MAX *(voir Modicon Quantum avec Unity, Modules réseau Ethernet, Manuel utilisateur)* ● Codes d'erreur EtherNet TCP/IP *(voir Modicon Quantum avec Unity, Modules réseau Ethernet, Manuel utilisateur)*
WRITE_REG	W_WARN_OUT_OF_RANGE	F	30 110	16#759E	Paramètre hors limites
WRITE_REG	E_EFB_NOT_STATE_RAM_4X	F	-30 531	16#88BD	Variable non affectée à la zone %MW (4x)
WRITE_REG	E_EFB_MSTR_ERROR	F	-30 191	16#8A11	Erreur de communication MSTR
WRITE_REG	-	F	8 195	16#2003	Valeur affichée dans le mot d'état (apparaît avec W_WARN_OUT_OF_RANGE)

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
WRITE_REG	-	F	8 206	16#200E	Valeur affichée dans le mot d'état (apparaît avec E_EFB_NOT_STATE_RAM_4X)
WRITE_REG	-	F	-	-	Voir les tableaux suivants : ● Codes d'erreur Modbus Plus et EtherNet SY/MAX *(voir Modicon Quantum avec Unity, Modules réseau Ethernet, Manuel utilisateur)* ● Codes d'erreur spécifiques SY/MAX *(voir Modicon Quantum avec Unity, Modules réseau Ethernet, Manuel utilisateur)* ● Codes d'erreur EtherNet TCP/IP *(voir Modicon Quantum avec Unity, Modules réseau Ethernet, Manuel utilisateur)*

Tableau des codes d'erreur pour la bibliothèque de gestion des E/S

Introduction

Les tableaux présentés dans cette section répertorient les codes et les valeurs d'erreur générés pour les EFB de la bibliothèque de gestion des E/S.

Configuration des E/S analogiques

Tableau des codes et valeurs d'erreur générés pour les EFB de la famille `Configuration des E/S analogiques`.

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
I_FILTER	E_EFB_NOT_CONFIGURED	F	-30188	16#8A14	La configuration EFB ne correspond pas à la configuration matérielle
I_SET	E_EFB_USER_ERROR_1	F	-30200	16#8A08	L'entrée IN_REG n'est pas associée au numéro d'un mot d'entrée (%IW)
I_SET	E_EFB_USER_ERROR_2	F	-30201	16#8A07	L'entrée IN_REG est associée à un numéro non valide d'un mot d'entrée (%IW)
I_SET	E_EFB_USER_ERROR_3	F	-30202	16#8A06	MN_RAW MX_RAW
I_SET	E_EFB_USER_ERROR_4	F	-30203	16#8A05	Valeur inconnue de MN_PHYS
I_SET	E_EFB_USER_ERROR_5	F	-30204	16#8A04	Valeur inconnue de MX_PHYS
I_SET	E_EFB_USER_ERROR_11	F	-30210	16#89FE	ST_REG n'est pas saisi
I_SET	E_EFB_USER_ERROR_12	F	-30211	16#89FD	ST_REG est trop grand
I_SET	E_EFB_USER_ERROR_13	F	-30212	16#89FC	ST_CH n'est pas saisi
O_FILTER	E_EFB_NOT_CONFIGURED	F	-30188	16#8A14	La configuration EFB ne correspond pas à la configuration matérielle
O_SET	E_EFB_USER_ERROR_1	F	-30200	16#8A08	L'entrée OUT_REG n'est pas associée au numéro d'un mot de sortie (%MW)
O_SET	E_EFB_USER_ERROR_2	F	-30201	16#8A07	L'entrée OUT_REG est associée à un numéro non valide d'un mot de sortie (%MW)

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
O_SET	E_EFB_USER_ERROR_3	F	-30202	16#8A06	MN_RAW MX_RAW
O_SET	E_EFB_USER_ERROR_4	F	-30203	16#8A05	Valeur inconnue de MN_PHYS
O_SET	E_EFB_USER_ERROR_5	F	-30204	16#8A04	Valeur inconnue de MX_PHYS
O_SET	E_EFB_USER_ERROR_11	F	-30210	16#89FE	ST_REG n'est pas saisi
O_SET	E_EFB_USER_ERROR_12	F	-30211	16#89FD	ST_REG est trop grand
O_SET	E_EFB_USER_ERROR_13	F	-30212	16#89FC	ST_CH n'est pas saisi

Affichage d'E/S analogiques

Tableau des codes et valeurs d'erreur générés pour les EFB de la famille `Affichage des E/S analogiques`.

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
I_NORM	E_EFB_NEG_OVER_RANGE	F	-30187	16#8A15	Dépassement négatif
I_NORM	E_EFB_NOT_CONFIGURED	F	-30188	16#8A14	La configuration EFB ne correspond pas à la configuration matérielle
I_NORM_ WARN	E_EFB_NO_WARNING_STATUS _AVAILABLE	F	-30189	16#8A13	Le module ne fournit aucun état d'avertissement
I_NORM_ WARN	E_EFB_POS_OVER_RANGE	F	-30186	16#8A16	Dépassement positif
I_NORM_ WARN	E_EFB_NEG_OVER_RANGE	F	-30187	16#8A15	Dépassement négatif
I_NORM_ WARN	E_EFB_NOT_CONFIGURED	F	-30188	16#8A14	La configuration EFB ne correspond pas à la configuration matérielle
I_PHYS	E_EFB_NO_WARNING_STATUS _AVAILABLE	F	-30189	16#8A13	Le module ne fournit aucun état d'avertissement
I_PHYS	E_INPUT_VALUE_OUT_OF_ RANGE	F	-30183	16#8A19	Valeur d'entrée hors limites
I_PHYS	E_EFB_NO_MEASURING_ RANGE	F	-30185	16#8A17	Erreur interne
I_PHYS	E_EFB_POS_OVER_RANGE	F	-30186	16#8A16	Dépassement positif

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
I_PHYS	E_EFB_NEG_OVER_RANGE	F	-30187	16#8A15	Dépassement négatif
I_PHYS	E_EFB_NOT_CONFIGURED	F	-30188	16#8A14	La configuration EFB ne correspond pas à la configuration matérielle
I_PHYS_ WARN	E_EFB_NO_WARNING_STATUS _AVAILABLE	F	-30189	16#8A13	Le module ne fournit aucun état d'avertissement
I_PHYS_ WARN	E_EFB_FILTER_SQRT_NOT_ AVAIL	F	-30195	16#8A0D	Le filtre SQRT n'est pas disponible
I_PHYS_ WARN	E_INPUT_VALUE_OUT_OF_ RANGE	F	-30183	16#8A19	Valeur d'entrée hors limites
I_PHYS_ WARN	E_EFB_NO_MEASURING_ RANGE	F	-30185	16#8A17	Erreur interne
I_PHYS_ WARN	E_EFB_POS_OVER_RANGE	F	-30186	16#8A16	Dépassement positif
I_PHYS_ WARN	E_EFB_NEG_OVER_RANGE	F	-30187	16#8A15	Dépassement négatif
I_PHYS_ WARN	E_EFB_NOT_CONFIGURED	F	-30188	16#8A14	La configuration EFB ne correspond pas à la configuration matérielle
I_RAW	E_EFB_OUT_OF_RANGE	F	-30192	16#8A10	Erreur interne : L'EFB a détecté une violation, par exemple l'écriture a dépassé les limites %MW (4x)
I_RAW	E_EFB_NOT_CONFIGURED	F	-30188	16#8A14	La configuration EFB ne correspond pas à la configuration matérielle
I_RAWSIM	E_EFB_NOT_CONFIGURED	F	-30188	16#8A14	La configuration EFB ne correspond pas à la configuration matérielle
I_SCALE	E_EFB_POS_OVER_RANGE	F	-30186	16#8A16	Dépassement positif
I_SCALE	E_EFB_NEG_OVER_RANGE	F	-30187	16#8A15	Dépassement négatif
I_SCALE	E_EFB_NOT_CONFIGURED	F	-30188	16#8A14	La configuration EFB ne correspond pas à la configuration matérielle
I_SCALE_ WARN	E_EFB_NO_WARNING_STATUS _AVAILABLE	F	-30189	16#8A13	Le module ne fournit aucun état d'avertissement

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
I_SCALE_ WARN	E_EFB_POS_OVER_RANGE	F	-30186	16#8A16	Dépassement positif
I_SCALE_ WARN	E_EFB_NEG_OVER_RANGE	F	-30187	16#8A15	Dépassement négatif
I_SCALE_ WARN	E_EFB_NOT_CONFIGURED	F	-30188	16#8A14	La configuration EFB ne correspond pas à la configuration matérielle
O_NORM	E_EFB_POS_OVER_RANGE	F	-30186	16#8A16	Dépassement positif
O_NORM	E_EFB_NEG_OVER_RANGE	F	-30187	16#8A15	Dépassement négatif
O_NORM	E_EFB_NOT_CONFIGURED	F	-30188	16#8A14	La configuration EFB ne correspond pas à la configuration matérielle
O_NORM_ WARN	E_EFB_POS_OVER_RANGE	F	-30186	16#8A16	Dépassement positif
O_NORM_ WARN	E_EFB_NEG_OVER_RANGE	F	-30187	16#8A15	Dépassement négatif
O_NORM_ WARN	E_EFB_NOT_CONFIGURED	F	-30188	16#8A14	La configuration EFB ne correspond pas à la configuration matérielle
O_PHYS	E_EFB_NO_MEASURING_ RANGE	F	-30185	16#8A17	Erreur interne
O_PHYS	E_EFB_POS_OVER_RANGE	F	-30186	16#8A16	Dépassement positif
O_PHYS	E_EFB_NEG_OVER_RANGE	F	-30187	16#8A15	Dépassement négatif
O_PHYS	E_EFB_NOT_CONFIGURED	F	-30188	16#8A14	La configuration EFB ne correspond pas à la configuration matérielle
O_PHYS_ WARN	E_EFB_NO_MEASURING_ RANGE	F	-30185	16#8A17	Erreur interne
O_PHYS_ WARN	E_EFB_POS_OVER_RANGE	F	-30186	16#8A16	Dépassement positif
O_PHYS_ WARN	E_EFB_NEG_OVER_RANGE	F	-30187	16#8A15	Dépassement négatif
O_PHYS_ WARN	E_EFB_NOT_CONFIGURED	F	-30188	16#8A14	La configuration EFB ne correspond pas à la configuration matérielle
O_RAW	E_EFB_NEG_OVER_RANGE	F	-30187	16#8A15	Dépassement négatif

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
O_RAW	E_EFB_NOT_CONFIGURED	F	-30188	16#8A14	La configuration EFB ne correspond pas à la configuration matérielle
O_SCALE	E_INPUT_VALUE_OUT_OF_RANGE	F	-30183	16#8A19	Valeur d'entrée hors limites
O_SCALE	E_EFB_POS_OVER_RANGE	F	-30186	16#8A16	Dépassement positif
O_SCALE	E_EFB_NEG_OVER_RANGE	F	-30187	16#8A15	Dépassement négatif
O_SCALE	E_EFB_NOT_CONFIGURED	F	-30188	16#8A14	La configuration EFB ne correspond pas à la configuration matérielle
O_SCALE_WARN	E_INPUT_VALUE_OUT_OF_RANGE	F	-30183	16#8A19	Valeur d'entrée hors limites
O_SCALE_WARN	E_EFB_POS_OVER_RANGE	F	-30186	16#8A16	Dépassement positif
O_SCALE_WARN	E_EFB_NEG_OVER_RANGE	F	-30187	16#8A15	Dépassement négatif
O_SCALE_WARN	E_EFB_NOT_CONFIGURED	F	-30188	16#8A14	La configuration EFB ne correspond pas à la configuration matérielle

E/S directe

Tableau des codes et valeurs d'erreur générés pour les EFB de la famille `E/S directe`.

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
IMIO_IN	-	F	0000	0000	La commande fonctionne correctement
IMIO_IN	-	F	8193	2001	Type de commande non valide (par exemple, le module d'E/S adressé n'est pas un module d'entrée)
IMIO_IN	-	F	8194	2002	Numéro de châssis ou d'emplacement non valide (l'affectation des E/S du configurateur ne contient aucune entrée de module pour cet emplacement)
IMIO_IN	-	F	8195	2003	Numéro d'emplacement non valide
IMIO_IN	-	F	-4095	F001	Le module ne fonctionne pas correctement
IMIO_OUT	-	F	0000	0000	La commande fonctionne correctement
IMIO_OUT	-	F	8193	2001	Type de commande non valide (par exemple, le module d'E/S adressé n'est pas un module d'entrée)
IMIO_OUT	-	F	8194	2002	Numéro de châssis ou d'emplacement non valide (l'affectation des E/S du configurateur ne contient aucune entrée de module pour cet emplacement)
IMIO_OUT	-	F	8195	2003	Numéro d'emplacement non valide
IMIO_OUT	-	F	-4095	F001	Le module ne fonctionne pas correctement

Configuration des E/S Quantum

Tableau des codes et valeurs d'erreur générés pour les EFB de la famille `Configuration des E/S Quantum`.

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
ACI030	E_EFB_NOT_CONFIGURED	F	-30188	16#8A14	La configuration EFB ne correspond pas à la configuration matérielle
ACI040	E_EFB_NOT_CONFIGURED	F	-30188	16#8A14	La configuration EFB ne correspond pas à la configuration matérielle
ACI040	E_EFB_CURRENT_MODE_NOT_ALLOWED	F	-30197	16#8A0B	Erreur du EFB : Mode courant non autorisé
ACO020	E_EFB_NOT_CONFIGURED	F	-30188	16#8A14	La configuration EFB ne correspond pas à la configuration matérielle
ACO130	E_EFB_NOT_CONFIGURED	F	-30188	16#8A14	La configuration EFB ne correspond pas à la configuration matérielle
ACO130	E_EFB_CURRENT_MODE_NOT_ALLOWED	F	-30197	16#8A0B	Erreur du EFB : Mode courant non autorisé
AII330	E_EFB_NOT_CONFIGURED	F	-30188	16#8A14	La configuration EFB ne correspond pas à la configuration matérielle
AII330	E_EFB_ILLEGAL_CONFIG_DATA	F	-30198	16#8A0A	Erreur du EFB : Données de configuration incorrectes
AII33010	E_EFB_NOT_CONFIGURED	F	-30188	16#8A14	La configuration EFB ne correspond pas à la configuration matérielle
AII33010	E_EFB_CURRENT_MODE_NOT_ALLOWED	F	-30197	16#8A0B	Erreur du EFB : Mode courant non autorisé
AIO330	E_EFB_NOT_CONFIGURED	F	-30188	16#8A14	La configuration EFB ne correspond pas à la configuration matérielle
AIO330	E_EFB_CURRENT_MODE_NOT_ALLOWED	F	-30197	16#8A0B	Erreur du EFB : Mode courant non autorisé
AMM090	E_EFB_NOT_CONFIGURED	F	-30188	16#8A14	La configuration EFB ne correspond pas à la configuration matérielle

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
ARI030	E_EFB_NOT_CONFIGURED	F	-30188	16#8A14	La configuration EFB ne correspond pas à la configuration matérielle
ARI030	E_EFB_ILLEGAL_CONFIG_DATA	F	-30198	16#8A0A	Erreur du EFB : Données de configuration incorrectes
ATI030	E_EFB_NOT_CONFIGURED	F	-30188	16#8A14	La configuration EFB ne correspond pas à la configuration matérielle
AVI030	E_EFB_NOT_CONFIGURED	F	-30188	16#8A14	La configuration EFB ne correspond pas à la configuration matérielle
AVO020	E_EFB_NOT_CONFIGURED	F	-30188	16#8A14	La configuration EFB ne correspond pas à la configuration matérielle
DROP	E_EFB_NOT_CONFIGURED	F	-30188	16#8A14	La configuration EFB ne correspond pas à la configuration matérielle
ERT_854_10	ES_WRONG_SLOT	F	20480	16#5000	-
ERT_854_10	E_WRONG_SLOT	F	-30215	16#89F9	Défini comme E_EFB_USER_ERROR_16
ERT_854_10	ES_HEALTHBIT	F	24576	16#6000	-
ERT_854_10	E_HEALTHBIT	F	-30216	16#89F8	Défini comme E_EFB_USER_ERROR_17
ERT_854_10	ES_TIMEOUT	F	32768	16#8000	-
ERT_854_10	E_TIMEOUT	F	-30210	16#89FE	Défini comme E_EFB_USER_ERROR_11
ERT_854_10	E_ERT_BASIC - valeurs	F	-30199	16#8A09	Défini comme E_EFB_USER_ERROR_1 + 1
ERT_854_10	E_WRONG_ANSW	F	-30211	16#89FD	Défini comme E_EFB_USER_ERROR_12
ERT_854_10	ES_CBUF_OFLOW	F	28672	16#7000	-
ERT_854_10	E_CBUF_OFLOW	F	-30217	16#89F7	Défini comme E_EFB_USER_ERROR_18

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
ERT_854_10	ES_WRONG_PAKET	F	8192	16#2000	-
ERT_854_10	E_WRONG_PAKET	F	-30212	16#89FC	Défini comme E_EFB_USER_ERROR_13
ERT_854_10	ES_WRONG_FELD	F	12288	16#3000	-
ERT_854_10	E_WRONG_FELD	F	-30213	16#89FB	Défini comme E_EFB_USER_ERROR_14
QUANTUM	E_EFB_NOT_CONFIGURED	F	-30188	16#8A14	La configuration EFB ne correspond pas à la configuration matérielle
QUANTUM	E_EFB_UNKNOWN_DROP	F	-30190	16#8A12	Station d'E/S inconnue / aucune affectation des E/S Quantum
XBE	E_EFB_NOT_CONFIGURED	F	-30188	16#8A14	La configuration EFB ne correspond pas à la configuration matérielle
XBE	E_EFB_UNKNOWN_DROP	F	-30190	16#8A12	Station d'E/S inconnue / aucune affectation des E/S Quantum
XDROP	E_EFB_NOT_CONFIGURED	F	-30188	16#8A14	La configuration EFB ne correspond pas à la configuration matérielle

NOTE : Pour plus de détails concernant le `ERT_854_10`, reportez-vous à la description ERT_854_10 *(voir Quantum avec Unity Pro, Module de consignation d'état 140 ERT 854 10, Manuel utilisateur)* dans la bibliothèque de gestion des E/S.

Tableaux des codes d'erreur pour la bibliothèque CONT_CTL

Introduction

Les tableaux ci-dessous répertorient les codes d'erreur et les valeurs générés pour les EFB de la bibliothèque CONT_CTL.

Condition

Tableau des codes et valeurs d'erreur générés pour les EFB de la famille `Condition`.

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
DTIME	W_WARN_OUT_OF_RANGE	T	30 110	16#759E	Paramètre hors limites
DTIME	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
DTIME	Valeurs de mot d'état	T/F	-	-	Pour plus de détails concernant le mot d'état `DTIME`, reportez-vous à la description de DTIME *(voir Unity Pro, Régulation, Bibliothèque de blocs)*
INTEGRATOR	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
INTEGRATOR	E_ERR_IB_MAX_MIN	F	-30 102	16#8A6A	YMAX < YMIN
INTEGRATOR	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
LAG_FILTER	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
LAG_FILTER	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
LDLG	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
LDLG	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
FIL	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
FIL	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
MFLOW	W_WARN_OUT_OF_RANGE	T	30 110	16#759E	Paramètre hors limites
MFLOW	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
MFLOW	Valeurs de mot d'état	T/F	-	-	Pour plus de détails concernant le mot d'état `MFLOW`, reportez-vous à la description de MFLOW *(voir Unity Pro, Régulation, Bibliothèque de blocs)*
QDTIME	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
SCALING	E_ERR_NULL_INPUT_SCALE	F	-30 121	16#8A57	Echelle d'entrée nulle : les limites max. et min. doivent être différentes
SCALING	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
SCALING	Valeurs de mot d'état	T/F	-	-	Pour plus de détails concernant le mot d'état `SCALING`, reportez-vous à la description de SCALING *(voir Unity Pro, Régulation, Bibliothèque de blocs)*
TOTALIZER	W_WARN_OUT_OF_RANGE	T	30 110	16#759E	Paramètre hors limites

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
TOTALIZER	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
TOTALIZER	W_WARN_TOTALIZER_CTER_MAX	T	30 113	16#75A1	La valeur maximale du compteur a été atteinte
TOTALIZER	Valeurs de mot d'état	T/F	-	-	Pour plus de détails concernant le mot d'état `TOTALIZER`, reportez-vous à la description de TOTALIZER *(voir Unity Pro, Régulation, Bibliothèque de blocs)*
VEL_LIM	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
VEL_LIM	E_ERR_AB1_MAX_MIN	F	-30 101	16#8A6B	YMAX < YMIN
VEL_LIM	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*

Automate

Tableau des codes et valeurs d'erreur générés pour les EFB de la famille `Automate`.

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
AUTOTUNE	W_WARN_OUT_OF_RANGE	T	30 110	16#759E	Paramètre hors limites
AUTOTUNE	E_ERR_NULL_INPUT_SCALE	F	-30 121	16#8A57	Echelle d'entrée nulle : les limites max. et min. doivent être différentes
AUTOTUNE	W_WARN_AUTOTUNE_ FAILED	T	30 111	16#759F	AUTOTUNE a échoué
AUTOTUNE	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
AUTOTUNE	E_ERR_AUTOTUNE_ID_ UNKNOWN	F	-30 120	16#8A58	L'EFB optimisé n'est pas autorisé ici ou n'a pas encore été appelé
AUTOTUNE	Valeurs de mot d'état	T/F	-	-	Pour plus de détails concernant le mot d'état `AUTOTUNE`, reportez-vous à la description de AUTOTUNE *(voir Unity Pro, Régulation, Bibliothèque de blocs)*
PI_B	W_WARN_OUT_OF_RANGE	T	30 110	16#759E	Paramètre hors limites
PI_B	E_ERR_NULL_INPUT_SCALE	F	-30 121	16#8A57	Echelle d'entrée nulle : les limites max. et min. doivent être différentes
PI_B	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
PI_B	Valeurs de mot d'état	T/F	-	-	Pour plus de détails concernant le mot d'état `PI_B`, reportez-vous à la description de PI_B *(voir Unity Pro, Régulation, Bibliothèque de blocs)*
PIDFF	W_WARN_OUT_OF_RANGE	T	30 110	16#759E	Paramètre hors limites

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
PIDFF	E_ERR_NULL_INPUT_SCALE	F	-30 121	16#8A57	Echelle d'entrée nulle : les limites max. et min. doivent être différentes
PIDFF	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
PIDFF	Valeurs de mot d'état	T/F	-	-	Pour plus de détails concernant le mot d'état `PIDFF`, reportez-vous à la description de PIDFF *(voir Unity Pro, Régulation, Bibliothèque de blocs)*
SAMPLETM	E_EFB_SAMPLE_TIME_ OVERFLOW	F	-30 184	16#8A18	Erreur interne
STEP2	W_WARN_OUT_OF_RANGE	T	30 110	16#759E	Paramètre hors limites
STEP2	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
STEP2	Valeurs de mot d'état	T/F	-	-	Pour plus de détails concernant le mot d'état `STEP2`, référez-vous à la description de STEP2 *(voir Unity Pro, Régulation, Bibliothèque de blocs)*
STEP3	W_WARN_OUT_OF_RANGE	T	30 110	16#759E	Paramètre hors limites
STEP3	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
STEP3	Valeurs de mot d'état	T/F	-	-	Pour plus de détails concernant le mot d'état `STEP3`, référez-vous à la description de STEP3 *(voir Unity Pro, Régulation, Bibliothèque de blocs)*

Mathématiques

Tableau des codes et valeurs d'erreur générés pour les EFB de la famille `Mathématiques`.

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
COMP_DB	W_WARN_OUT_OF_RANGE	T	30 110	16#759E	Paramètre hors limites
COMP_DB	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
K_SQRT	W_WARN_OUT_OF_RANGE	T	30 110	16#759E	Paramètre hors limites
K_SQRT	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
MULDIV_W	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
SUM_W	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*

Mesure

Tableau des codes et valeurs d'erreur générés pour les EFB de la famille `Mesure`.

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
AVGMV	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
AVGMV	W_WARN_AVGMV	T	30 108	16#759C	AVGMV : N doit être <= 50
AVGMV	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
AVGMV_K	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
AVGMV_K	W_WARN_AVGMV_K	T	30 109	16#759D	AVGMV_K : N doit être <= 10000
AVGMV_K	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
DEAD_ZONE	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
DEAD_ZONE	E_ERR_DZONE	F	-30 119	16#8A59	DZONE : DZ doit être >= 0
DEAD_ZONE	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
LOOKUP_TABLE1	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
LOOKUP_TABLE1	E_ERR_POLY_ANZAHL	F	-30 107	16#8A65	Nombre impair d'entrées
LOOKUP_TABLE1	E_ERR_POLY_FOLGE	F	-30 108	16#8A64	point de base x(i) <= x(i-1)
LOOKUP_TABLE1	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*

Traitement en sortie

Tableau des codes et valeurs d'erreur générés pour les EFB de la famille `Traitement en sortie`.

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
MS	W_WARN_OUT_OF_RANGE	T	30 110	16#759E	Paramètre hors limites
MS	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
MS	Valeurs de mot d'état	T/F	-	-	Pour plus de détails concernant le mot d'état `MS`, reportez-vous à la description de MS *(voir Unity Pro, Régulation, Bibliothèque de blocs)*
PWM1	WAF_PBM_TMINMAX	F	-30 113	16#8A5F	t_min < t_max
PWM1	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
SERVO	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
SERVO	Valeurs de mot d'état	T/F	-	-	Pour plus de détails concernant le mot d'état `SERVO`, reportez-vous à la description de SERVO *(voir Unity Pro, Régulation, Bibliothèque de blocs)*
SPLRG	W_WARN_OUT_OF_RANGE	T	30 110	16#759E	Paramètre hors limites
SPLRG	E_ERR_NULL_INPUT_SCALE	F	-30 121	16#8A57	Echelle d'entrée nulle : les limites max. et min. doivent être différentes
SPLRG	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
SPLRG	Valeurs de mot d'état	T/F	-	-	Pour plus de détails concernant le mot d'état `SPLRG`, reportez-vous à la description de SPLRG *(voir Unity Pro, Régulation, Bibliothèque de blocs)*

Gestion de consigne

Tableau des codes et valeurs d'erreur générés pour les EFB de la famille `Gestion de consigne`.

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
RAMP	W_WARN_OUT_OF_RANGE	T	30 110	16#759E	Paramètre hors limites
RAMP	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
RAMP	Valeurs de mot d'état	T/F	-	-	Pour plus de détails concernant le mot d'état `RAMP`, reportez-vous à la description de RAMP *(voir Unity Pro, Régulation, Bibliothèque de blocs)*
RATIO	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
RATIO	Valeurs de mot d'état	T/F	-	-	Pour plus de détails concernant le mot d'état `RATIO`, reportez-vous à la description de RATIO *(voir Unity Pro, Régulation, Bibliothèque de blocs)*
SP_SEL	W_WARN_OUT_OF_RANGE	T	30 110	16#759E	Paramètre hors limites
SP_SEL	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
SP_SEL	Valeurs de mot d'état	T/F	-	-	Pour plus de détails concernant le mot d'état `SP_SEL`, reportez-vous à la description de SP_SEL *(voir Unity Pro, Régulation, Bibliothèque de blocs)*

Tableaux des codes d'erreur pour la bibliothèque de mouvements

Introduction

Les tableaux présentés dans cette section répertorient les codes et les valeurs d'erreur générés pour les EFB de la bibliothèque de mouvements.

Démarrage MMF

Tableau des codes et valeurs d'erreur générés pour les EFB de la famille `Démarrage MMF`.

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
CFG_CP_F	BAD_REVISION	F	-30 200	16#8A08	défini comme E_EFB_USER_ERROR_1
CFG_CP_F	MMF_BAD_4X	T	9 010	16#2332	-
CFG_CP_F	MMF_ABORT_SUB	T	7 004	16#1B5C	erreur de liaison SubNum/SubNumEcho
CFG_CP_V	BAD_REVISION	F	-30 200	16#8A08	défini comme E_EFB_USER_ERROR_1
CFG_CP_V	MMF_BAD_4X	T	9 010	16#2332	-
CFG_CP_V	MMF_ABORT_SUB	T	7 004	16#1B5C	erreur de liaison SubNum/SubNumEcho
CFG_CS	BAD_REVISION	F	-30 200	16#8A08	défini comme E_EFB_USER_ERROR_1
CFG_CS	MMF_ABORT_SUB	T	7 004	16#1B5C	erreur de liaison SubNum/SubNumEcho
CFG_FS	BAD_REVISION	F	-30 200	16#8A08	défini comme E_EFB_USER_ERROR_1
CFG_FS	MMF_ABORT_SUB	T	7 004	16#1B5C	erreur de liaison SubNum/SubNumEcho
CFG_IA	BAD_REVISION	F	-30 200	16#8A08	défini comme E_EFB_USER_ERROR_1
CFG_IA	MMF_ABORT_SUB	T	7 004	16#1B5C	erreur de liaison SubNum/SubNumEcho
CFG_RA	BAD_REVISION	F	-30 200	16#8A08	défini comme E_EFB_USER_ERROR_1
CFG_RA	MMF_ABORT_SUB	T	7 004	16#1B5C	erreur de liaison SubNum/SubNumEcho

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
CFG_SA	BAD_REVISION	F	-30 200	16#8A08	défini comme E_EFB_USER_ERROR_1
CFG_SA	MMF_ABORT_SUB	T	7 004	16#1B5C	erreur de liaison SubNum/SubNumEcho
DRV_DNLD	BAD_REVISION	F	-30 200	16#8A08	défini comme E_EFB_USER_ERROR_1
DRV_DNLD	MMF_ABORT_SUB	T	7 004	16#1B5C	erreur de liaison SubNum/SubNumEcho
DRV_UPLD	BAD_REVISION	F	-30 200	16#8A08	défini comme E_EFB_USER_ERROR_1
DRV_UPLD	MMF_ABORT_SUB	T	7 004	16#1B5C	erreur de liaison SubNum/SubNumEcho
IDN_CHK	BAD_REVISION	F	-30 200	16#8A08	défini comme E_EFB_USER_ERROR_1
IDN_CHK	MMF_ABORT_SUB	T	7 004	16#1B5C	erreur de liaison SubNum/SubNumEcho
IDN_XFER	BAD_REVISION	F	-30 200	16#8A08	défini comme E_EFB_USER_ERROR_1
IDN_XFER	MMF_ABORT_SUB	T	7 004	16#1B5C	erreur de liaison SubNum/SubNumEcho
MMF_BITS	BAD_REVISION	F	-30 200	16#8A08	défini comme E_EFB_USER_ERROR_1
MMF_ESUB	BAD_REVISION	F	-30 200	16#8A08	défini comme E_EFB_USER_ERROR_1
MMF_ESUB	MMF_ABORT_SUB	T	7 004	16#1B5C	erreur de liaison SubNum/SubNumEcho
MMF_IDNX	BAD_REVISION	F	-30 200	16#8A08	défini comme E_EFB_USER_ERROR_1
MMF_IDNX	MMF_ABORT_SUB	T	7 004	16#1B5C	erreur de liaison SubNum/SubNumEcho
MMF_JOG	BAD_REVISION	F	-30 200	16#8A08	défini comme E_EFB_USER_ERROR_1
MMF_JOG	MMF_ABORT_SUB	T	7 004	16#1B5C	erreur de liaison SubNum/SubNumEcho
MMF_JOG	MMF_SUB_TIMEOUT	T	7 005	16#1B5D	Le sous-programme ne s'est pas terminé à temps

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
MMF_MOVE	BAD_REVISION	F	-30 200	16#8A08	défini comme E_EFB_USER_ERROR_1
MMF_MOVE	MMF_ABORT_SUB	T	7 004	16#1B5C	erreur de liaison SubNum/SubNumEcho
MMF_RST	BAD_REVISION	F	-30 200	16#8A08	défini comme E_EFB_USER_ERROR_1
MMF_SUB	BAD_REVISION	F	-30 200	16#8A08	défini comme E_EFB_USER_ERROR_1
MMF_SUB	MMF_ABORT_SUB	T	7 004	16#1B5C	erreur de liaison SubNum/SubNumEcho
MMF_USUB	BAD_REVISION	F	-30 200	16#8A08	défini comme E_EFB_USER_ERROR_1
MMF_USUB	MMF_ABORT_SUB	T	7 004	16#1B5C	erreur de liaison SubNum/SubNumEcho

NOTE : Pour plus de détails concernant les codes et les valeurs d'erreur MMF, reportez-vous à la section Rapports d'erreurs et de défauts *(voir Unity Pro, Commande de mouvement, Bibliothèque de blocs)* dans la bibliothèque de mouvements.

Tableaux des codes d'erreur pour la bibliothèque d'obsolescences

Introduction

Les tableaux présentés dans cette section répertorient les codes et les valeurs d'erreur générés pour les EFB de la bibliothèque d'obsolescences.

CLC

Tableau des codes et valeurs d'erreur générés pour les EFB de la famille `CLC`.

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
DELAY	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
INTEGRATOR1	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
INTEGRATOR1	E_ERR_IB_MAX_MIN	F	-30 102	16#8A6A	YMAX < YMIN
INTEGRATOR1	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
LAG1	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
LAG1	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
LEAD_LAG1	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
LEAD_LAG1	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
LIMV	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
LIMV	E_ERR_AB1_MAX_MIN	F	-30 101	16#8A6B	YMAX < YMIN
LIMV	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
PI1	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
PI1	E_ERR_PI_MAX_MIN	F	-30 103	16#8A69	YMAX < YMIN
PI1	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
PID1	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
PID1	E_ERR_PID_MAX_MIN	F	-30 104	16#8A68	YMAX < YMIN
PID1	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
PIDP1	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
PIDP1	E_ERR_PID_MAX_MIN	F	-30 104	16#8A68	YMAX < YMIN
PIDP1	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
SMOOTH_RATE	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
SMOOTH_RATE	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
THREE_STEP_CON1	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
THREE_STEP_CON1	W_WARN_DSR_TN	T	30 101	16#7595	TN = 0
THREE_STEP_CON1	W_WARN_DSR_TSN	T	30 102	16#7596	TSN = 0
THREE_STEP_CON1	W_WARN_DSR_KP	T	30 103	16#7597	KP <= 0
THREE_STEP_CON1	E_ERR_DSR_HYS	F	-30 105	16#8A67	2 * |UZ| < |HYS|
THREE_STEP_CON1	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
THREEPOINT_CON1	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
THREEPOINT_CON1	W_WARN_ZDR_XRR	F	30 105	16#7599	DR : XRR < -100 ou XRR > 100
THREEPOINT_CON1	W_WARN_ZDR_T1T2	F	30 104	16#7598	T2 > T1
THREEPOINT_CON1	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
THREEPOINT_CON1	E_ERR_ZDR_HYS	F	-30 106	16#8A66	2 * |UZ| < |HYS|
TWOPOINT_CON1	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
TWOPOINT_CON1	W_WARN_ZDR_XRR	F	30 105	16#7599	DR : XRR < -100 ou XRR > 100
TWOPOINT_CON1	W_WARN_ZDR_T1T2	F	30 104	16#7598	T2 > T1
TWOPOINT_CON1	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
TWOPOINT_CON1	E_ERR_ZDR_HYS	F	-30 106	16#8A66	2 * |UZ| < |HYS|

CLC_PRO

Tableau des codes et valeurs d'erreur générés pour les EFB de la famille CLC_PRO.

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
ALIM	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
ALIM	WAF_AB2_VMAX	F	-30 111	16#8A61	vmax <= 0
ALIM	WAF_AB2_BMAX	F	-30 112	16#8A60	bmax <= 0
ALIM	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
COMP_PID	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
COMP_PID	WAF_KPID_KUZ	F	-30 110	16#8A62	gain_red < 0 ou gain_red > 1
COMP_PID	WAF_KPID_OGUG	F	-30 104	16#8A68	YMAX < YMIN
COMP_PID	WAF_KPID_UZ	F	-30 109	16#8A63	db < 0
COMP_PID	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
DEADTIME	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
DERIV	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
DERIV	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
FGEN	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
FGEN	WAF_SIG_TV_MAX	F	-30 116	16#8A5C	t_acc > t_rise / 2
FGEN	WAF_SIG_TH_MAX	F	-30 117	16#8A5B	t_rise trop élevé
FGEN	WAF_SIG_TA_MAX	T	30 106	16#759A	t_off >= halfperiod
FGEN	WAF_SIG_T1_MIN	T	30 107	16#759B	t_max <= t_min
FGEN	WAF_SIG_FKT	F	-30 118	16#8A5A	func_no <= 0 ou func_no > 8
FGEN	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
INTEG	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
INTEG	E_ERR_IB_MAX_MIN	F	-30 102	16#8A6A	YMAX < YMIN
INTEG	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
LAG	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
LAG	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
LAG2	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
LAG2	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
LEAD_LAG	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
LEAD_LAG	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
PCON2	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
PCON2	W_WARN_ZDR_XRR	T	30 105	16#7599	DR : XRR < -100 ou XRR > 100
PCON2	W_WARN_ZDR_T1T2	T	30 104	16#7598	T2 > T1
PCON2	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
PCON2	E_ERR_ZDR_HYS	F	-30 106	16#8A66	2 * |UZ| < |HYS|
PCON3	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
PCON3	W_WARN_ZDR_XRR	T	30 105	16#7599	DR : XRR < -100 ou XRR > 100
PCON3	W_WARN_ZDR_T1T2	T	30 104	16#7598	T2 > T1
PCON3	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
PCON3	E_ERR_ZDR_HYS	F	-30 106	16#8A66	2 * |UZ| < |HYS|
PD_OR_PI	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
PD_OR_PI	WAF_PDPI_OG_UG	F	-30 103	16#8A69	YMAX < YMIN
PD_OR_PI	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
PDM	PDM_TMAX_TMIN	F	-30 115	16#8A5D	t_max <= t_min
PDM	PDM_OG_UG	F	-30 114	16#8A69	|pos_up_x| > |pos_lo_x| ou |neg_up_x| > |neg_lo_x|
PDM	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
PI	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
PI	E_ERR_PI_MAX_MIN	F	-30 103	16#8A69	YMAX < YMIN
PI	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
PID	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
PID	E_ERR_PID_MAX_MIN	F	-30 104	16#8A68	YMAX < YMIN
PID	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
PID_P	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
PID_P	E_ERR_PID_MAX_MIN	F	-30 104	16#8A68	YMAX < YMIN
PID_P	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
PIP	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
PIP	E_ERR_PI_MAX_MIN	F	-30 103	16#8A69	YMAX < YMIN
PIP	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
PPI	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
PPI	E_ERR_PI_MAX_MIN	F	-30 103	16#8A69	YMAX < YMIN
PPI	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
PWM	WAF_PBM_TMINMAX	F	-30 113	16#8A5F	t_min < t_max
PWM	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
QPWM	WAF_PBM_TMINMAX	F	-30 113	16#8A5F	t_min < t_max
QPWM	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
SCON3	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
SCON3	W_WARN_DSR_TN	T	30 101	16#7595	TN = 0
SCON3	W_WARN_DSR_TSN	T	30 102	16#7596	TSN = 0

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
SCON3	W_WARN_DSR_KP	T	30 103	16#7597	KP <= 0
SCON3	E_ERR_DSR_HYS	F	-30 105	16#8A67	2 * |UZ| < |HYS|
SCON3	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*
VLIM	E_ERR_DEN	F	-30 152	16#8A38	Nombre à virgule flottante incorrect
VLIM	E_ERR_AB1_MAX_MIN	F	-30 101	16#8A6B	YMAX < YMIN
VLIM	FP_ERROR	F	-	-	Voir tableau des *Erreurs courantes relatives aux valeurs en virgule flottante, page 655*

Extension/Compatibilité

Tableau des codes et valeurs d'erreur générés pour les EFB de la famille `Extension/Compatibilité`.

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
AKF_TA	E_AKFEFB_TIMEBASE_IS_ZERO	F	-30 482	16#88EE	Base temporelle zéro
AKF_TE	E_AKFEFB_TIMEBASE_IS_ZERO	F	-30 482	16#88EE	Base temporelle zéro
AKF_TI	E_AKFEFB_TIMEBASE_IS_ZERO	F	-30 482	16#88EE	Base temporelle zéro
AKF_TS	E_AKFEFB_TIMEBASE_IS_ZERO	F	-30 482	16#88EE	Base temporelle zéro
AKF_TV	E_AKFEFB_TIMEBASE_IS_ZERO	F	-30 482	16#88EE	Base temporelle zéro
FIFO	E_INPUT_VALUE_OUT_OF_RANGE	F	-30 183	16#8A19	Valeur d'entrée hors limites
GET_3X	E_INPUT_VALUE_OUT_OF_RANGE	F	-30 183	16#8A19	Valeur d'entrée hors limites
GET_4X	E_INPUT_VALUE_OUT_OF_RANGE	F	-30 183	16#8A19	Valeur d'entrée hors limites
GET_BIT	E_INPUT_VALUE_OUT_OF_RANGE	F	-30 183	16#8A19	Valeur d'entrée hors limites
IEC_BMDI	E_EFB_USER_ERROR_1	F	-30 200	16#8A08	Type de registre invalide pour la valeur d'entrée (SourceTable).

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
IEC_BMDI	E_EFB_USER_ERROR_2	F	-30 201	16#8A07	Sélection par le décalage d'entrée (OffsetInSourceTable) d'une adresse en dehors des limites acceptables.
IEC_BMDI	E_EFB_USER_ERROR_3	F	-30 202	16#8A06	Le décalage d'entrée (OFF_IN) n'est pas égal à 1 ni un multiple de 16+1.
IEC_BMDI	E_EFB_USER_ERROR_4	F	-30 203	16#8A05	Type de registre invalide pour la valeur de sortie (DestinationTable).
IEC_BMDI	E_EFB_USER_ERROR_5	F	-30 204	16#8A04	Sélection par le décalage de sortie (OffsetInDestinationTable) d'une adresse en dehors des limites acceptables.
IEC_BMDI	E_EFB_USER_ERROR_6	F	-30 205	16#8A03	Le décalage de sortie (OffsetInDestinationTable) n'est pas égal à 1 ni un multiple de 16+1.
IEC_BMDI	E_EFB_USER_ERROR_7	F	-30 206	16#8A02	La valeur de (NumberOfElements) est 0.
IEC_BMDI	E_EFB_USER_ERROR_8	F	-30 207	16#8A01	La valeur de (NumberOfElements) traite plus de 1 600 bits.
IEC_BMDI	E_EFB_USER_ERROR_9	F	-30 208	16#8A00	La valeur de (NumberOfElements) traite plus de 100 mots.
IEC_BMDI	E_EFB_USER_ERROR_10	F	-30 209	16#89FF	La valeur de (NumberOfElements) sélectionne une adresse source en dehors des limites acceptables.

Nom EFB	Code d'erreur	Etat ENO en cas d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
IEC_BMDI	E_EFB_USER_ERROR_11	F	-30 210	16#89FE	La valeur de (NumberOfElements) sélectionne une adresse cible en dehors des limites acceptables.
IEC_BMDI	E_EFB_USER_ERROR_12	F	-30 211	16#89FD	La valeur de (NumberOfElements) n'est pas un multiple de 16.
IEC_BMDI	E_EFB_USER_ERROR_13	F	-30 212	16#89FC	Avertissement : Dépassement d'adresse des adresses d'entrée et de sortie.
LIFO	E_INPUT_VALUE_OUT_OF_RANGE	F	-30 183	16#8A19	Valeur d'entrée hors limites
PUT_4X	E_INPUT_VALUE_OUT_OF_RANGE	F	-30 183	16#8A19	Valeur d'entrée hors limites
MUX_DINTARR_125	E_SELECTOR_OUT_OF_RANGE	F	-30 175	16#8A21	Sélecteur hors limites
SET_BIT	E_INPUT_VALUE_OUT_OF_RANGE	F	-30 183	16#8A19	Valeur d'entrée hors limites

Erreurs courantes relatives aux valeurs en virgule flottante

Introduction

Le tableau ci-dessous répertorie les codes d'erreur et les valeurs générés par des erreurs relatives aux valeurs en virgule flottante.

Erreurs courantes relatives aux valeurs en virgule flottante

Tableau des erreurs courantes relatives aux valeurs en virgule flottante

Codes d'erreur	Valeur d'erreur (format décimal)	Valeur d'erreur (format hexadécimal)	Description de l'erreur
FP_ERROR	-30 150	16#8A3A	Valeur de base (n'apparaît pas comme une valeur d'erreur)
E_FP_STATUS_FAILED_IE	-30 151	16#8A39	Opération incorrecte de virgule flottante
E_FP_STATUS_FAILED_DE	-30 152	16#8A38	L'opérande n'est pas un nombre de type REAL valide
E_FP_STATUS_FAILED_ZE	-30 154	16#8A36	Division par zéro incorrecte
E_FP_STATUS_FAILED_ZE_IE	-30 155	16#8A35	Opération incorrecte de virgule flottante / Division par zéro
E_FP_STATUS_FAILED_OE	-30 158	16#8A32	Dépassement de virgule flottante
E_FP_STATUS_FAILED_OE_IE	-30 159	16#8A31	Opération incorrecte de virgule flottante / Dépassement
E_FP_STATUS_FAILED_OE_ZE	-30 162	16#8A2E	Dépassement de virgule flottante / Division par zéro
E_FP_STATUS_FAILED_OE_ZE_IE	-30 163	16#8A2D	Opération incorrecte de virgule flottante / Dépassement / Division par zéro
E_FP_NOT_COMPARABLE	-30 166	16#8A2A	Erreur interne

Conformité CEI

B

Vue d'ensemble

Ce chapitre contient les tableaux de conformité associés à la norme CEI 61131-3.

Contenu de ce chapitre

Ce chapitre contient les sous-chapitres suivants :

B.1 Informations générales relatives à la norme CEI 61131-3

Informations générales relatives à la conformité CEI 61131-3

Présentation

La norme CEI 61131-3 (voir sous-alinéa 1.4) définit la syntaxe et la sémantique d'une gamme unifiée de langages de programmation destinés aux automates programmables. Parmi ces langages figurent deux langages textuels, IL (Instruction List - Liste d'instructions) et ST (Structured Text - Littéral structuré), et deux langages graphiques, LD (Ladder Diagram - Langage à contacts) et FBD (Function Block Diagram - Langage en blocs fonctionnels).

Cette norme définit également les éléments du langage SFC (Sequential Function Chart - Diagramme fonctionnel en séquence) qui permettent de structurer l'organisation interne des programmes et blocs fonction des automates programmables. Des éléments de configuration permettent également de prendre en charge l'installation des programmes dans les systèmes d'automates programmables.

NOTE : Unity Pro utilise les acronymes anglais pour désigner les langages de programmation.

Des fonctionnalités permettent aussi de faciliter la communication entre les automates programmables et les autres composants des systèmes automatisés.

Conformité de Unity Pro avec la norme CEI 61131-3

La version actuelle du système de programmation Unity Pro prend en charge un ensemble conforme d'éléments de langage définis dans la norme.

Signification du terme "conformité" dans ce contexte :

- Conformément à la norme, l'installateur d'un système de programmation CEI peut choisir ou retirer des fonctionnalités de langage spécifiques, voire des langages complets, dans les tables des fonctionnalités qui font partie intégrante des exigences du système. Un système dit conforme doit mettre en œuvre les fonctionnalités choisies en respectant les exigences de la norme.
- En outre, l'installateur peut utiliser les éléments de langage de programmation définis dans un environnement de programmation interactif. La norme stipule clairement qu'elle ne couvre pas la définition de tels environnements. De ce fait, l'installateur dispose d'un certain degré de liberté et peut proposer des procédures de traitement et de présentation optimisées pour des éléments de langage spécifiques.
- Unity Pro laisse ainsi une certaine liberté à l'utilisateur qui peut créer des "projets" pour traiter de manière combinée les éléments "Configuration" et "Ressource" du langage CEI. L'utilisateur bénéficie également d'une certaine souplesse concernant les mécanismes de traitement des déclarations de variable ou d'instanciation de bloc fonction.

Tableaux de conformité CEI

Conformément à la norme, les fonctionnalités prises en charge et les autres informations de mise en œuvre spécifiques sont fournies dans la déclaration de conformité et les tableaux associés (voir sections suivantes).

B.2 Tableaux de conformité CEI

Vue d'ensemble

Ce système respecte les exigences de la norme CEI 61131-3 en matière de langages et fonctionnalités (voir tableaux ci-après).

Contenu de ce sous-chapitre

Ce sous-chapitre contient les sujets suivants :

Eléments communs

Eléments communs

Tableau de conformité CEI des éléments communs :

N° de tableau	N° de fonctionnalité	Description de la fonctionnalité
1	2	Caractères en minuscules
	3a	Signe nombre (#)
	4a	Signe dollar ($)
	5a	Barre verticale (|)
2	1	Caractères en majuscules et nombres
	2	Caractères en minuscules et majuscules, nombres, caractères de soulignement intégrés
	3	Caractères en minuscules et majuscules, nombres, caractères de soulignement de début ou intégrés
3	1	Commentaires
3a	1	Directives pragma
4	1	Valeur littérale entier
	2	Valeur littérale réelle
	3	Valeur littérale réelle avec exposant
	4	Valeur littérale en base 2
	5	Valeur littérale en base 8
	6	Valeur littérale en base 16
	7	Booléen 0 et 1
	8	Booléen `FALSE` et `TRUE`
	9	Valeur littérale typée
5	1	Chaîne de caractères un bit
	3	Valeur littérale chaîne typée un bit
6	2	Signe dollar
	3	Apostrophe
	4	Retour à la ligne
	5	Nouvelle ligne
	6	Saut de page
	7	Retour chariot
	8	Tabulation
	9	Guillemet

N° de tableau	N° de fonctionnalité	Description de la fonctionnalité
7	1a	Valeur littérale de durée sans caractère de soulignement : préfixe court
	1b	Préfixe long
	2a	Valeur littérale de durée avec caractère de soulignement : préfixe court
	2b	Préfixe long
8	1	Valeur littérale de date (préfixe long)
	2	Valeur littérale de date (préfixe court)
	3	Valeur littérale d'heure (préfixe long)
	4	Valeur littérale d'heure (préfixe court)
	5	Valeur littérale de date et heure (préfixe long)
	5	Valeur littérale de date et heure (préfixe court)
10	1	Type de données `BOOL`
	3	Type de données `INT`
	4	Type de données `DINT`
	7	Type de données `UINT`
	8	Type de données `UDINT`
	10	Type de données `REAL`
	12	Type de données `TIME`
	13	Type de données `DATE`
	14	Type de données `TIME_OF_DAY` ou `TOD`
	15	Type de données `DATE_AND_TIME` ou `DT`
	16	Type de données `CHAINE`
	17	Type de données `OCTET`
	18	Type de données `MOT`
	19	Type de données `MOTD`
12	4	Type de données Tableau
	5	Types de données structurées
14	4	Initialisation de types de données tableau
	6	Initialisation de types de données structurées dérivées

N° de tableau	N° de fonctionnalité	Description de la fonctionnalité
15	1	Emplacement entrée
	2	Emplacement sortie
	3	Emplacement mémoire
	4	Taille d'un bit (préfixe X)
	5	Taille d'un bit (pas de préfixe)
	7	Taille d'un mot (16 bits)
	8	Taille d'un mot double (32 bits)
	9	Taille d'un mot long (quad - 64 bits)
17	3	Déclaration d'emplacements de variables symboliques (*Remarque 5, page 670*)
	4	Affectation d'emplacements de tableau (*Remarque 5, page 670*)
	5	Affectation automatique en mémoire de variables symboliques
	6	Déclaration de tableaux (*Remarque 11, page 672*)
	7	Déclaration de tableaux mémorisés (*Remarque 11, page 672*)
	8	Déclaration de variables structurées
18	1	Initialisation de variables directement représentées (*Remarque 11, page 672*)
	3	Affectation d'emplacements et de valeurs initiales de variables symboliques
	4	Affectation et initialisation d'emplacements de tableau
	5	Initialisation de variables symboliques
	6	Initialisation de tableaux (*Remarque 11, page 672*)
	7	Déclaration et initialisation de tableaux mémorisés (*Remarque 11, page 672*)
	8	Initialisation de variables structurées
	9	Initialisation de constantes
	10	Initialisation d'instances de bloc fonction
19	1	Entrée inversée
	2	Sortie inversée
19a	1	Appel formel de fonction/bloc fonction
	2	Appel informel de fonction/bloc fonction

N° de tableau	N° de fonctionnalité	Description de la fonctionnalité
20	1	Utilisation des paramètres EN et ENO indiquée dans le langage LD
	2	Utilisation sans paramètres EN et ENO indiquée dans le langage FBD
20a	1	Déclaration de variables d'E/S (textuel)
	2	Déclaration de variables d'E/S (graphique)
	3	Liaison graphique de variable d'E/S à des variables différentes (graphique)
21	1	Fonctions surchargées
	2	Fonctions typées
22	1	*_TO_** (*Remarque 1, page 669*)
	2	TRUNC (*Remarque 2, page 669*)
	3	*_BCD_TO_** (*Remarque 3, page 670*)
	4	**_TO_BCD_* (*Remarque 3, page 670*)
23	1	Fonction ABS
	2	Fonction SQRT
	3	Fonction LN
	4	Fonction LOG
	5	Fonction EXP
	6	Fonction SIN
	7	Fonction COS
	8	Fonction TAN
	9	Fonction ASIN
	10	Fonction ACOS
	11	Fonction ATAN
24	12	Fonction ADD
	13	Fonction MUL
	14	Fonction SUB
	15	Fonction DIV
	16	Fonction MOD
	17	Fonction EXPT
	18	Fonction MOVE

N° de tableau	N° de fonctionnalité	Description de la fonctionnalité
25	1	Fonction SHL
	2	Fonction SHR
	3	Fonction ROR
	4	Fonction ROL
26	5	Fonction AND
	6	Fonction OR
	7	Fonction XOR
	8	Fonction NOT
27	1	Fonction SEL
	2a	Fonction MAX
	2b	Fonction MIN
	3	Fonction LIMIT
	4	Fonction MUX
28	5	Fonction GT
	6	Fonction GE
	7	Fonction EQ
	8	Fonction LE
	9	Fonction LT
	10	Fonction NE
29	1	Fonction LEN (*Remarque 4, page 670*)
	2	Fonction LEFT (*Remarque 4, page 670*)
	3	Fonction RIGHT (*Remarque 4, page 670*)
	4	Fonction MID (*Remarque 4, page 670*)
	6	Fonction INSERT (*Remarque 4, page 670*)
	7	Fonction DELETE (*Remarque 4, page 670*)
	8	Fonction REPLACE (*Remarque 4, page 670*)
	9	Fonction FIND (*Remarque 4, page 670*)

N° de tableau	N° de fonctionnalité	Description de la fonctionnalité
30	1a	Fonction ADD (*Remarque 6, page 672*)
	1b	Fonction ADD_TIME
	2b	Fonction ADD_TOD_TIME
	3b	Fonction ADD_DT_TIME
	4a	Fonction SUB (*Remarque 6, page 672*)
	4b	Fonction SUB_TIME
	5b	Fonction SUB_DATE_DATE
	6b	Fonction SUB_TOD_TIME
	7b	Fonction SUB_TOD_TOD
	8b	Fonction SUB_DT_TIME
	9b	Fonction SUB_DT_DT
	10a	Fonction MUL (*Remarque 6, page 672*)
	10b	Fonction MULTIME
	11a	Fonction DIV (*Remarque 6, page 672*)
	11b	Fonction DIVTIME
33	1a	Qualificatif RETAIN pour variables internes (*Remarque 11, page 672*)
	2a	Qualificatif RETAIN pour variables de sortie (*Remarque 11, page 672*)
	2b	Qualificatif RETAIN pour variables d'entrée (*Remarque 11, page 672*)
	3a	Qualificatif RETAIN pour blocs fonction internes (*Remarque 11, page 672*)
	4a	Déclaration de variables VAR_IN_OUT (textuel)
	4b	Déclaration et utilisation de variables VAR_IN_OUT (graphique)
	4c	Déclaration de variables VAR_IN_OUT avec affectation à différentes variables (graphique)
34	1	Bloc fonction bistable (réglage dominant)
	2	Bloc fonction bistable (réarmement dominant)
35	1	Détecteur front montant
	2	Détecteur front descendant

N° de tableau	N° de fonctionnalité	Description de la fonctionnalité
36	1a	Bloc fonction CTU (compteur)
	1b	Bloc fonction CTU_DINT
	1d	Bloc fonction CTU_UDINT
	2a	Bloc fonction CTD (décompteur)
	2b	Bloc fonction CTD_DINT
	2d	Bloc fonction CTD_UDINT
	3a	Bloc fonction CTUD (compteur-décompteur)
	3b	Bloc fonction CTUD_DINT
	3d	Bloc fonction CTUD_UDINT
37	1	Bloc fonction TP (impulsion)
	2a	Bloc fonction TON (délai à l'activation)
	3a	Bloc fonction TOF (délai à la désactivation)
39	19	Utilisation de variables directement représentées
40	1	Etape et étape initiale (forme graphique avec liaisons directes)
	3a	Indicateur d'étape (forme générale)
	4	Temps écoulé pour l'étape (forme générale)
41	7	Utilisation du nom de transition
	7a	Condition de transition associée via le nom de transition dans le langage LD
	7b	Condition de transition associée via le nom de transition dans le langage FBD
	7c	Condition de transition associée via le nom de transition dans le langage IL
	7d	Condition de transition associée via le nom de transition dans le langage ST
42	1	Toute variable booléenne déclarée dans un bloc VAR ou VAR_OUTPUT, ou son équivalent graphique, peut être une action
	2l	Déclaration graphique d'action dans le langage LD
	2f	Déclaration graphique d'action dans le langage FBD
	3s	Déclaration textuelle d'action dans le langage ST
	3i	Déclaration textuelle d'action dans le langage IL

N° de tableau	N° de fonctionnalité	Description de la fonctionnalité
43	1	Bloc action voisin, de manière physique ou logique, de l'étape (*Remarque 7, page 672*)
	2	Blocs action concaténés voisins, de manière physique ou logique, de l'étape (*Remarque 8, page 672*)
44	1	Qualificatif d'action dans bloc action pris en charge
	2	Nom d'action dans bloc action pris en charge
45	1	`None` - pas de qualificatif
	2	Qualificatif `N`
	3	Qualificatif `R`
	4	Qualificatif `S`
	5	Qualificatif `L`
	6	Qualificatif `D`
	7	Qualificatif `P`
	9	Qualificatif `DS`
	11	Qualificatif `P1`
	12	Qualificatif `P0`
45a	2	Contrôle d'action sans "cycle final"
46	1	Séquence unique
	2a	Divergence de sélection de séquence : priorité des évaluations de transition de gauche à droite
	3	Convergence de sélection de séquence
	4	Séquences simultanées - divergence et convergence
	5a	Saut de séquence : priorité des évaluations de transition de gauche à droite
	6a	Boucle de séquence : priorité des évaluations de transition de gauche à droite
49	1	Construction `CONFIGURATION...END_CONFIGURATION` (*Remarque 12, page 672*)
	5a	Construction `TASK` périodique
	5b	Construction `TASK` non périodique
	6a	Construction `WITH` pour association `PROGRAM` dans `TASK` (*Remarque 9, page 672*)
	6c	Déclaration `PROGRAM` sans association `TASK` (*Remarque 10, page 672*)
50	5a	Ordonnancement non préemptif (*Remarque 13, page 672*)
	5b	Ordonnancement préemptif (*Remarque 14, page 672*)

Remarque 1

Liste des fonctions de conversion de type :

- BOOL_TO_BYTE, BOOL_TO_DINT, BOOL_TO_INT, BOOL_TO_REAL, BOOL_TO_TIME, BOOL_TO_UDINT, BOOL_TO_UINT, BOOL_TO_WORD, BOOL_TO_DWORD
- BYTE_TO_BOOL, BYTE_TO_DINT, BYTE_TO_INT, BYTE_TO_REAL, BYTE_TO_TIME, BYTE_TO_UDINT, BYTE_TO_UINT, BYTE_TO_WORD, BYTE_TO_DWORD, BYTE_TO_BIT
- DINT_TO_BOOL, DINT_TO_BYTE, DINT_TO_INT, DINT_TO_REAL, DINT_TO_TIME, DINT_TO_UDINT, DINT_TO_UINT, DINT_TO_WORD, DINT_TO_DWORD, DINT_TO_DBCD, DINT_TO_STRING
- INT_TO_BOOL, INT_TO_BYTE, INT_TO_DINT, INT_TO_REAL, INT_TO_TIME, INT_TO_UDINT, INT_TO_UINT, INT_TO_WORD, INT_TO_BCD, INT_TO_DBCD, INT_TO_DWORD, INT_TO_STRING
- REAL_TO_BOOL, REAL_TO_BYTE, REAL_TO_DINT, REAL_TO_INT, REAL_TO_TIME, REAL_TO_UDINT, REAL_TO_UINT, REAL_TO_WORD, REAL_TO_DWORD, REAL_TO_STRING
- TIME_TO_BOOL, TIME_TO_BYTE, TIME_TO_DINT, TIME_TO_INT, TIME_TO_REAL, TIME_TO_UDINT, TIME_TO_UINT, TIME_TO_WORD, TIME_TO_DWORD, TIME_TO_STRING
- UDINT_TO_BOOL, UDINT_TO_BYTE, UDINT_TO_DINT, UDINT_TO_INT, UDINT_TO_REAL, UDINT_TO_TIME, UDINT_TO_UINT, UDINT_TO_WORD, UDINT_TO_DWORD
- UINT_TO_BOOL, UINT_TO_BYTE, UINT_TO_DINT, UINT_TO_INT, UINT_TO_REAL, UINT_TO_TIME, UINT_TO_UDINT, UINT_TO_WORD, UINT_TO_DWORD
- WORD_TO_BOOL, WORD_TO_BYTE, WORD_TO_DINT, WORD_TO_INT, WORD_TO_REAL, WORD_TO_TIME, WORD_TO_UDINT, WORD_TO_UINT, WORD_TO_BIT, WORD_TO_DWORD
- DWORD_TO_BOOL, DWORD_TO_BYTE, DWORD_TO_DINT, DWORD_TO_INT, DWORD_TO_REAL, DWORD_TO_TIME, DWORD_TO_UDINT, DWORD_TO_UINT, DWORD_TO_BIT

Les effets de chaque conversion sont décrits dans l'aide fournie avec la bibliothèque de base.

Remarque 2

Liste de types pour la fonction de troncature :

- REAL_TRUNC_DINT, REAL_TRUNC_INT, REAL_TRUNC_UDINT, REAL_TRUNC_UINT

Les effets de chaque conversion sont décrits dans l'aide fournie avec la bibliothèque de base.

Remarque 3

Liste de types pour la fonction de conversion BCD :

- BCD_TO_INT, DBCD_TO_INT, DBCD_TO_DINT

Liste de types pour la fonction de conversion BCD :

- INT_TO_BCD, INT_TO_DBCD, DINT_TO_DBCD

Les effets de chaque conversion sont décrits dans l'aide fournie avec la bibliothèque de base.

Remarque 4

Liste de types pour la fonction chaîne :

- LEN_INT, LEFT_INT, RIGHT_INT, MID_INT, INSERT_INT, DELETE_INT, REPLACE_INT, FIND_INT

Remarque 5

Une variable peut être affectée à une variable directement représentée lorsque le type est strictement identique.

Par exemple, une variable de type INT peut uniquement être affectée à une variable directement représentée de type INT.

Il existe cependant une exception : Pour les variables de mémoire de mot interne (%MW<i>), mot linéaire (%IW<i>) et mot constante (%KW<i>), vous pouvez utiliser n'importe quel type de variable déclarée.

Affectations autorisées :

	Syntaxe	**Type de données**	**Types de variable autorisés**
Bit interne	%M<i> ou %MX<i>	EBOOL	EBOOL ARRAY [..] OF EBOOL
Mot interne	%MW<i>	INT	Tous les types sont autorisés à l'exception de : • EBOOL • ARRAY [..] OF EBOOL
Mot double interne	%MD<i>	DINT	Aucune affectation, car chevauchement entre %MW<i>, %MD<i> et %MF<i>
Réel interne	%MF<i>	REAL	Aucune affectation, car chevauchement entre %MW<i>, %MD<i> et %MF<i>
Mot constante	%KW<i>	INT	Tous les types sont autorisés à l'exception de : • EBOOL • ARRAY [..] OF EBOOL

	Syntaxe	Type de données	Types de variable autorisés
Mot double constante	%KD<i>	DINT	Aucune affectation, car chevauchement entre %KW<i>, %KD<i> et %KF<i> Ce type de variables existe uniquement sur les automates Premium.
Réel constante	%KF<i>	REAL	Aucune affectation, car chevauchement entre %KW<i>, %KD<i> et %KF<i> Ce type de variables existe uniquement sur les automates Premium.
Bit système	%S<i> ou %SX<i>	EBOOL	EBOOL
Mot système	%SW<i>	INT	INT
Mot double système	%SD<i>	DINT	DINT
Bit linéaire	%I<i>	EBOOL	EBOOL ARRAY [..] OF EBOOL Ce type de variables existe uniquement sur les automates Quantum.
Mot linéaire	%IW<i>	INT	Tous les types sont autorisés à l'exception de : • EBOOL • ARRAY [..] OF EBOOL Ce type de variables existe uniquement sur les automates Quantum.
Mot commun	%NWi.j.k	INT	INT
Variables topologiques	%I..., %Q..., ...	...	Type identique (Sur certains modules d'E/S numériques, il est possible d'affecter des tableaux EBOOL aux objets %IX<topo> et %QX<topo>.)
Bit extrait	%MWi.j, ...	BOOL	BOOL

Remarque 6

Uniquement l'opérateur "+" (pour ADD), "-" (pour SUB), "*" (pour MUL) ou "/" (pour DIV) dans le langage ST.

Remarque 7

Cette fonctionnalité apparaît uniquement dans l'"affichage étendu" du diagramme.

Remarque 8

Cette fonctionnalité apparaît dans l'"affichage étendu" du diagramme non pas sous la forme de blocs concaténés, mais sous la forme d'une liste déroulante de noms d'action avec qualificatifs associés dans un symbole de bloc.

Remarque 9

Il existe uniquement une injection de l'instance PROGRAM dans TASK. Le format textuel est remplacé par une boîte de dialogue de propriétés.

Remarque 10

Le format textuel est remplacé par une boîte de dialogue de propriétés.

Remarque 11

Toutes les variables sont mémorisées (qualificatif `RETAIN` utilisé de manière implicite dans les déclarations de variables).

Remarque 12

Le format textuel est remplacé par le Navigateur de projet.

Remarque 13

L'utilisateur peut obtenir un comportement non préemptif à l'aide de l'instruction Mask-IT. Les paramètres `MASKEVT` (masquage global des événements) et `UNMASKEVT` (démasquage global des événements) sont disponibles dans les fonctions système de la bibliothèque.

Remarque 14

Par défaut, le système multitâche est préemptif.

Eléments de langage IL

Eléments de langage IL

Tableau de conformité CEI des éléments de langage IL :

N° de tableau	N° de fonctionnalité	Description de la fonctionnalité
51b	1	Expression entre parenthèses commençant par un opérateur explicite
51b	2	Expression entre parenthèses (forme courte)
52	1	Opérateur `LD` (avec modificateur "`N`")
	2	Opérateur `ST` (avec modificateur "`N`")
	3	Opérateur `S`, `R`
	4	Opérateur `AND` (avec modificateurs "`(`", "`N`")
	6	Opérateur `OR` (avec modificateurs "`(`", "`N`")
	7	Opérateur `XOR` (avec modificateurs "`(`", "`N`")
	7a	Opérateur `NOT`
	8	Opérateur `ADD` (avec modificateur "`(`")
	9	Opérateur `SUB` (avec modificateur "`(`")
	10	Opérateur `MUL` (avec modificateur "`(`")
	11	Opérateur `DIV` (avec modificateur "`(`")
	11a	Opérateur `MOD` (avec modificateur "`(`")
	12	Opérateur `GT` (avec modificateur "`(`")
	13	Opérateur `GE` (avec modificateur "`(`")
	14	Opérateur `EQ` (avec modificateur "`(`")
	15	Opérateur `NE` (avec modificateur "`(`")
	16	Opérateur `LE` (avec modificateur "`(`")
	17	Opérateur `LT` (avec modificateur "`(`")
	18	Opérateur `JMP` (avec modificateurs "`C`", "`N`")
	19	Opérateur `CAL` (avec modificateurs "`C`", "`N`")
	20	Opérateur `RET` (avec modificateurs "`C`", "`N`") (*Remarque, page 674*)
	21	`)` (opération d'évaluation différée)

N° de tableau	N° de fonctionnalité	Description de la fonctionnalité
53	1a	CAL pour bloc fonction avec liste d'arguments informelle
	1b	CAL pour bloc fonction avec liste d'arguments formelle
	2	CAL pour bloc fonction avec chargement/stockage d'arguments
	4	Appel de fonction avec liste d'arguments formelle
	5	Appel de fonction avec liste d'arguments informelle

Remarque

Dans bloc DFB uniquement.

Eléments de langage ST

Eléments de langage ST

Tableau de conformité CEI des éléments de langage ST :

N° de tableau	N° de fonctionnalité	Description de la fonctionnalité
55	1	Mise entre parenthèses (expression)
	2	Evaluation de fonction : Nomfonction(listeArguments)
	3	Elévation à une puissance : `**`
	4	Négation : `-`
	5	Complément : `NOT`
	6	Multiplication : `*`
	7	Division : `/`
	8	Modulo : `MOD`
	9	Addition : `+`
	10	Soustraction : `-`
	11	Comparaison : `<`, `>`, `<=`, `>=`
	12	Egalité : `=`
	13	Inégalité : `<>`
	14	Booléen ET : `&`
	15	Booléen ET : `AND`
	16	Booléen OU exclusif : `XOR`
	17	Booléen OU : `OR`

N° de tableau	N° de fonctionnalité	Description de la fonctionnalité
56	1	Affectation
	2	Appel de bloc fonction et utilisation sortie bloc fonction
	3	Instruction `RETURN` (*Remarque, page 676*)
	4	Instruction `IF`
	5	Instruction `CASE`
	6	Instruction `FOR`
	7	Instruction `WHILE`
	8	Instruction `REPEAT`
	9	Instruction `EXIT`
	10	Instruction vide

Remarque

Dans bloc DFB uniquement.

Eléments graphiques communs

Eléments graphiques communs

Tableau de conformité CEI des éléments graphiques communs :

N° de tableau	N° de fonctionnalité	Description de la fonctionnalité
57	2	Lignes horizontales : graphique ou semi-graphique
	4	Lignes verticales : graphique ou semi-graphique
	6	Connexion horizontale/verticale : graphique ou semi-graphique
	8	Franchissements de lignes sans connexion : graphique ou semi-graphique
	10	Coins connectés et non connectés : graphique ou semi-graphique
	12	Blocs avec lignes de connexion : graphique ou semi-graphique
58	1	Saut inconditionnel : langage FBD
	2	Saut inconditionnel : langage LD
	3	Saut conditionnel : langage FBD
	4	Saut conditionnel : langage LD
	5	Retour conditionnel : langage LD (*Remarque, page 677*)
	6	Retour conditionnel : langage FBD (*Remarque, page 677*)
	7	Retour inconditionnel depuis fonction ou bloc fonction (*Remarque, page 677*)
	8	Retour inconditionnel : langage LD (*Remarque, page 677*)

Remarque

Dans bloc DFB uniquement.

Eléments de langage LD

Eléments de langage LD

Tableau de conformité CEI des éléments de langage LD :

N° de tableau	N° de fonctionnalité	Description de la fonctionnalité
59	1	Barre d'alimentation gauche
	2	Barre d'alimentation droite
60	1	Liaison horizontale
	2	Liaison verticale
61	1	Contact à fermeture (barre verticale) (*Remarque, page 678*)
	3	Contact à ouverture (barre verticale) (*Remarque, page 678*)
	5	Contact de détection de transition positive (barre verticale) (*Remarque, page 678*)
	7	Contact de détection de transition négative (barre verticale) (*Remarque, page 678*)
62	1	Bobine
	2	Bobine inverse
	3	Bobine SET (mémorisation)
	4	Bobine RESET (déclenchement)
	8	Bobine de détection de transition positive
	9	Bobine de détection de transition négative

Remarque

Représentation graphique seulement.

Paramètres dépendants de la mise en oeuvre

Paramètres dépendants de la mise en œuvre

Tableau de conformité CEI des paramètres dépendants de la mise en œuvre :

Paramètres	Limites et comportement
Longueur maximum des identificateurs	32 caractères
Longueur maximum des commentaires	Dans Unity Pro : 1 024 caractères par objet d'éditeur. Importation : limité par les contraintes XML ou l'utilisation de chaînes UDB dans la couche persistante.
Syntaxe et sémantique des directives pragma	Unity V1.0 prend uniquement en charge 1 directive pragma, utilisée pour le convertisseur hérité : `{ ConvError (' texte erreur'); }` toute autre construction pragma est ignorée (un message d'avertissement est fourni)
Syntaxe et sémantique pour l'utilisation du caractère guillemet lorsqu'une application particulière prend en charge la fonctionnalité n° 4 du tableau 5, mais pas la fonctionnalité n° 2.	(la fonctionnalité n° 2 du tableau 5 est prise en charge)
Plage de valeurs et précision de la représentation pour les variables de type `TIME`, `DATE`, `TIME_OF_DAY` et `DATE_AND_TIME`	pour `TIME` : `t#0ms .. t#4294967295ms` (=`t#49D_17H_2M_47S_295MS`) pour `DATE` : `D#1990-01-01 .. D#2099-12-31` pour `TOD` : `TOD#00:00:00 .. TOD#23:59:59`
Précision de la représentation des secondes pour les types `TIME`, `TIME_OF_DAY` et `DATE_AND_TIME`	`TIME` : précision d'1 ms `TIME_OF_DAY` : précision d'1 s
Nombre maximum de valeurs énumérées	Non applicable
Nombre maximum d'indices de tableau	6
Taille de tableau maximum	64 Ko
Nombre maximum d'éléments de structure	pas de limite
Taille de structure maximum	64 Ko
Plage maximum de valeurs d'indice	plage `DINT`
Nombre maximum de niveaux dans structures imbriquées	10
Longueur maximum par défaut des variables `CHAINE` et `WSTRING`	16 caractères

Paramètres	Limites et comportement
Longueur maximum autorisée des variables CHAINE et WSTRING	64 Ko
Nombre maximum de niveaux hiérarchiques Affectation logique ou physique	Premium : affectation physique (5 niveaux) Quantum : affectation logique (1 niveau)
Nombre maximum d'entrées pour les fonctions extensibles	Le nombre maximum de paramètres d'entrée (paramètres d'E/S inclus) est égal à 32. Le nombre maximum de paramètres de sortie (paramètres d'E/S inclus) est aussi égal à 32. Le nombre maximum de paramètres d'entrée pour les fonctions extensibles est donc égal à (32 - nombre de paramètres d'entrée - nombre de paramètres d'E/S). Le nombre maximum de paramètres de sortie pour les fonctions extensibles est donc égal à (32 - nombre de paramètres de sortie - nombre de paramètres d'E/S).
Effets des conversions de type de données sur la précision	Reportez-vous à l'aide en ligne.
Conditions d'erreur lors des conversions de type de données	Les conditions d'erreur sont décrites dans l'aide en ligne. Globalement, %S18 est activé pour les erreurs de dépassement. ENO est également activé. Le résultat dépend de la fonction utilisée.
Précision des fonctions numériques	Traitement ou émulation de virgule flottante INTEL
Effets des conversions entre des types de données temporelles et d'autres types de données non définis dans le tableau 30	Reportez-vous à l'aide en ligne.
Nombre maximum d'indications et instanciations de blocs fonction	Seulement limité par la taille maximum d'une section.
Affectation de variables d'entrée pour blocs fonction lorsque EN est sur FALSE	Pas d'affectation

Paramètres	Limites et comportement
Pvmin, Pvmax de compteurs	Compteurs base INT : • Pvmin=-32768 (0x8000) • Pvmax=32767 (0x7FFF) Compteurs base UINT : • Pvmin=0 (0x0) • Pvmax=65535 (0xFFFF) Compteurs base DINT : • Pvmin= -2147483648 (0x80000000) • Pvmax=2147483647 (0x7FFFFFFF) Compteurs base UDINT : • Pvmin=0 (0x0) • Pvmax=4294967295 (0xFFFFFFFF)
Effet du changement de la valeur d'une entrée PT pendant une opération de temporisation	Les nouvelles valeurs PT sont immédiatement prises en compte, même lorsqu'une opération de temporisation est en cours.
Limites liées à la taille des programmes	Dépend du type d'automate et de la mémoire
Précision au niveau du temps écoulé associé à une étape	10 ms
Nombre maximum d'étapes par SFC	1 024 étapes par section SFC
Nombre maximum de transitions par SFC et par étape	Limité par la zone disponible pour la saisie des étapes/transitions et par le nombre maximum d'étapes par section SFC (1 024 étapes). 32 transitions par étape. Limité par la zone disponible pour la saisie des branches Divergence en OU/Divergence en ET, 32 lignes maximum.
Nombre maximum de blocs action par étape	20
Accès à l'équivalent fonctionnel des sorties Q ou A	Non applicable
Temps de franchissement d'une transition	Dépendant de la transition toujours < 100 micro-secondes
Profondeur maximum des constructions de divergence/convergence	32
Contenu des bibliothèques RESOURCE	Non applicable
Effet de l'utilisation de l'accès READ_WRITE sur les sorties de blocs fonction	Non applicable
Nombre maximum de tâches	Dépend du type d'automate. Nombre maximum sur l'automate le plus puissant : 9 tâches

Paramètres	**Limites et comportement**
Résolution d'intervalle de tâche	10 ms
Longueur maximum des expressions	Presque sans limite
Longueur maximum des instructions	Presque sans limite
Nombre maximum de sélections `CASE`	Presque sans limite
Valeur de la variable de contrôle après exécution d'une boucle `FOR`	Non défini
Limites liées à la topologie du réseau	Aucune restriction
Ordre d'évaluation des boucles de retour	Le bloc connecté à la variable de retour est exécuté en premier.

Conditions d'erreur

Conditions d'erreur

Tableau de conformité CEI des conditions d'erreur :

Conditions d'erreur	**Traitement (*Remarque, page 684*)**
Commentaires imbriqués	2) erreur signalée pendant la programmation
La valeur d'une variable dépasse la sous-plage définie	4) erreur signalée pendant l'exécution
Configuration manquante d'une indication d'adresse incomplète (notation "*")	Non applicable
Tentative, par une unité organisationnelle du programme, de modification d'une variable déclarée en tant que `CONSTANTE`	2) erreur signalée pendant la programmation
Utilisation incorrecte, dans des fonctions, de variables externes ou directement représentées	Non applicable
Une variable `VAR_IN_OUT` n'est pas "correctement affectée"	2) erreur signalée pendant la programmation
Erreurs de conversion de type	4) erreur signalée pendant l'exécution
Le résultat numérique dépasse la plage définie pour le type de données	4) erreur signalée pendant l'exécution
Division par zéro	4) erreur signalée pendant l'exécution
Types de données mixtes en entrée pour une fonction de sélection	2) erreur signalée pendant la programmation
Le résultat dépasse la plage définie pour le type de données	4) erreur signalée pendant l'exécution
Aucune valeur définie pour une variable d'E/S	2) erreur signalée pendant la programmation
Zéro ou plus d'une étape initiale dans le réseau SFC	3) erreur signalée pendant l'analyse/le chargement/la liaison
Le programme utilisateur tente de modifier l'état ou l'heure d'une étape	2) erreur signalée pendant la programmation
Effets de bord pendant l'évaluation d'une condition de transition	3) erreur signalée pendant l'analyse/le chargement/la liaison
Erreur de conflit au niveau du contrôle des actions	3) erreur signalée pendant l'analyse/le chargement/la liaison
Transitions sans ordre de priorité simultanément vraies dans une divergence de sélection	Non applicable
SFC non fiable ou inaccessible	3) erreur signalée pendant l'analyse/le chargement/la liaison

Conditions d'erreur	Traitement (*Remarque, page 684*)
Conflit de type de données dans `VAR_ACCESS`	Non applicable
Une tâche ne parvient pas à être ordonnancée ou délai d'exécution impossible à respecter	4) erreur signalée pendant l'exécution
Le résultat numérique dépasse la plage définie pour le type de données	4) erreur signalée pendant l'exécution
Les types de données du résultat courant et de l'opérande sont différents	2) erreur signalée pendant la programmation
Division par zéro	4) erreur signalée pendant l'exécution
Le résultat numérique dépasse la plage définie pour le type de données	4) erreur signalée pendant l'exécution
Type de données incorrect pour l'opération	4) erreur signalée pendant l'exécution
Retour de fonction sans valeur affectée	Non applicable
Echec de fin de l'itération	4) erreur signalée pendant l'exécution
Le même identificateur est utilisé comme libellé de connecteur et nom d'élément	Non applicable
Variable de retour non initialisée	1) erreur non signalée

Remarque

Identifications pour le traitement des conditions d'erreur (conformément à la norme CEI 61131-3, sous-alinéa 1.5.1, d) :

- 1) erreur non signalée
- 2) erreur signalée pendant la programmation
- 3) erreur signalée pendant l'analyse/le chargement/la liaison
- 4) erreur signalée pendant l'exécution

B.3 Extensions de la norme CEI 61131-3

Extensions de la norme CEI 61131-3, deuxième édition

Présentation

En plus des fonctionnalités CEI normalisées répertoriées dans les *(voir page 660)*, l'environnement de programmation Unity Pro a hérité d'un certain nombre de fonctionnalités issues de l'environnement de programmation PL7. Ces extensions sont fournies en option et peuvent être activées depuis une boîte de dialogue d'options. La boîte de dialogue et les fonctionnalités sont détaillées dans le chapitre *Données et langages (voir Unity Pro, Modes de marche,)* de l'aide en ligne.

La boîte de dialogue des options ne fait pas état d'une autre extension héritée à la fois des environnements de programmation PL7 et Concept : Unity Pro permet la construction de la fameuse "section" dans tous les langages de programmation et autorise ainsi la sous-division d'une unité organisationnelle de programme (POU - Program Organization Unit). Il devient alors possible de combiner plusieurs langages (sections FBD, sections SFC, par exemple) dans une unité POU. Lorsqu'elle est utilisée dans cet objectif, cette fonctionnalité constitue une extension de la syntaxe CEI. Une unité POU conforme doit être composée d'une seule section. Les sections ne créent pas de domaine d'application de nom distinct. Le domaine d'application de nom de tous les éléments de langage est le POU.

Objectif des sections

Les sections sont utilisées à différentes fins :

- Les sections permettent de sous-diviser des unités POU larges en fonction d'aspects fonctionnels. L'utilisateur a la possibilité de sous-diviser son unité POU en différentes parties significatives d'un point de vue fonctionnel. La liste de sections représente un tableau fonctionnel du contenu d'une unité POU large et autrement non structurée.
- Les sections permettent de sous-diviser des unités POU larges en fonction d'aspects graphiques. L'utilisateur a la possibilité de définir des sous-structures pour une unité POU large selon une présentation graphique propre. Il est parfaitement libre de créer des petites sections graphiques ou bien des grandes sections.
- La sous-division d'unités POU larges permet d'apporter rapidement des modifications en ligne : sous Unity Pro, la section est utilisée comme unité pour la modification en ligne. Lorsqu'une unité POU est modifiée à différents emplacements alors que le système est en cours d'exécution, toutes les sections affectées par ces modifications sont automatiquement chargées via une requête explicite.

- Les sections permettent de redéfinir l'ordre dans lequel certaines parties libellées d'une unité POU sont exécutées : le nom de la section représente le libellé de la partie de cette unité contenue dans la section. En organisant ces libellés, l'utilisateur peut gérer l'ordre d'exécution des parties.
- Avec les sections, il est possible d'utiliser différents langages en parallèle au sein de la même unité POU : cette fonctionnalité représente une extension majeure de la syntaxe CEI, cette dernière autorisant en effet un seul langage CEI dans une unité POU. Dans une unité conforme, les sections SFC doivent être utilisées pour gérer différents langages au sein d'une unité (chaque transition/action peut être formulée dans un langage qui lui est propre).

B.4 Syntaxe des langages textuels

Syntaxe des langages textuels

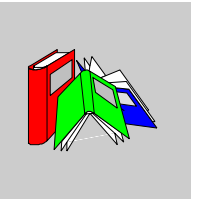
Description

L'environnement de programmation Unity Pro V1.0 ne prend pas encore en charge l'importation et l'exportation de fichiers texte conformes à la syntaxe complète des langages textuels, telle qu'elle est définie dans l'Annexe B de la norme CEI 61131-3, deuxième édition.

En revanche, la syntaxe textuelle des langages IL et ST, telle qu'elle est définie dans l'Annexe B.2 et B.3 de la norme CEI 61131-3, deuxième édition (productions associées de manière directe et indirecte à l'Annexe B.1 incluses), est prise en charge dans les sections en langage textuel.

Ces productions de syntaxe de l'Annexe B de la norme CEI 61131-3, deuxième édition qui appartiennent à des fonctionnalités non prises en charge par Unity Pro (cf. tableaux de conformité *(voir page 660)*) ne sont pas mises en œuvre.

Glossaire



0-9

%I

Selon le standard IEC, %I indique un objet langage de type entrée TOR.

%ID

Selon le standard IEC, %MW indique un objet langage de type double mot d'entrée.

Seuls les objets d'E/S permettent de localiser les instances de type (%MD<i>, %KD<i>, %QD, %ID, %MF<i>, %KF<i>, %QF, %IF) en utilisant leur adresse topologique (par exemple %MD0.6.0.11, %MF0.6.0.31).

%IF

Selon le standard IEC, %MW indique un objet langage de type réel d'entrée.

Seuls les objets d'E/S permettent de localiser les instances de type (%MD<i>, %KD<i>, %QD, %ID, %MF<i>, %KF<i>, %QF, %IF) en utilisant leur adresse topologique (par exemple %MD0.6.0.11, %MF0.6.0.31).

%IW

Selon le standard IEC, %IW indique un objet langage de type entrée analogique.

%KD

Selon le standard IEC, %MW indique un objet langage de type double mot constant.

Pour les automates Premium/Atrium, les instances de type double de données localisées (%MD<i>, %KD<i>) ou flottantes (%MF<i>, %KF<i>) doivent être localisées par un type d'entier (%MW<i>, %KW<i>). Seuls les objets d'E/S permettent de localiser les instances de type (%MD<i>, %KD<i>, %QD, %ID, %MF<i>, %KF<i>, %QF, %IF) en utilisant leur adresse topologique (par exemple %MD0.6.0.11, %MF0.6.0.31).

Pour les automates Modicon M340, les instances de type double de données localisées (%MD<i>, %KD<i>) ou flottantes (%MF<i>, %KF<i>) ne sont pas disponibles.

%KF

Selon le standard IEC, %MW indique un objet langage de type réel constant.

Pour les automates Premium/Atrium, les instances de type double de données localisées (%MD<i>, %KD<i>) ou flottantes (%MF<i>, %KF<i>) doivent être localisées par un type d'entier (%MW<i>, %KW<i>). Seuls les objets d'E/S permettent de localiser les instances de type (%MD<i>, %KD<i>, %QD, %ID, %MF<i>, %KF<i>, %QF, %IF) en utilisant leur adresse topologique (par exemple %MD0.6.0.11, %MF0.6.0.31).

Pour les automates Modicon M340, les instances de type double de données localisées (%MD<i>, %KD<i>) ou flottantes (%MF<i>, %KF<i>) ne sont pas disponibles.

%KW

Selon le standard IEC, %KW indique un objet langage de type mot constant.

Pour les automates Premium/Atrium, les instances de type double de données localisées (%MD<i>, %KD<i>) ou flottantes (%MF<i>, %KF<i>) doivent être localisées par un type d'entier (%MW<i>, %KW<i>). Seuls les objets d'E/S permettent de localiser les instances de type (%MD<i>, %KD<i>, %QD, %ID, %MF<i>, %KF<i>, %QF, %IF) en utilisant leur adresse topologique (par exemple %MD0.6.0.11, %MF0.6.0.31).

Pour les automates Modicon M340, les instances de type double de données localisées (%MD<i>, %KD<i>) ou flottantes (%MF<i>, %KF<i>) ne sont pas disponibles.

%M

Selon le standard IEC, %M indique un objet langage de type bit mémoire.

%MD

Selon le standard IEC, %MW indique un objet langage de type double mot mémoire.

Pour les automates Premium/Atrium, les instances de type double de données localisées (%MD<i>, %KD<i>) ou flottantes (%MF<i>, %KF<i>) doivent être localisées par un type d'entier (%MW<i>, %KW<i>). Seuls les objets d'E/S permettent de localiser les instances de type (%MD<i>, %KD<i>, %QD, %ID, %MF<i>, %KF<i>, %QF, %IF) en utilisant leur adresse topologique (par exemple %MD0.6.0.11, %MF0.6.0.31).

Pour les automates Modicon M340, les instances de type double de données localisées (%MD<i>, %KD<i>) ou flottantes (%MF<i>, %KF<i>) ne sont pas disponibles.

%MF

Selon le standard IEC, %MW indique un objet langage de type réel mémoire.

Pour les automates Premium/Atrium, les instances de type double de données localisées (%MD<i>, %KD<i>) ou flottantes (%MF<i>, %KF<i>) doivent être localisées par un type d'entier (%MW<i>, %KW<i>). Seuls les objets d'E/S permettent de localiser les instances de type (%MD<i>, %KD<i>, %QD, %ID, %MF<i>, %KF<i>, %QF, %IF) en utilisant leur adresse topologique (par exemple %MD0.6.0.11, %MF0.6.0.31).

Pour les automates Modicon M340, les instances de type double de données localisées (%MD<i>, %KD<i>) ou flottantes (%MF<i>, %KF<i>) ne sont pas disponibles.

%MW

Selon le standard IEC, %MW indique un objet langage de type mot mémoire.

Pour les automates Premium/Atrium, les instances de type double de données localisées (%MD<i>, %KD<i>) ou flottantes (%MF<i>, %KF<i>) doivent être localisées par un type d'entier (%MW<i>, %KW<i>). Seuls les objets d'E/S permettent de localiser les instances de type (%MD<i>, %KD<i>, %QD, %ID, %MF<i>, %KF<i>, %QF, %IF) en utilisant leur adresse topologique (par exemple %MD0.6.0.11, %MF0.6.0.31).

Pour les automates Modicon M340, les instances de type double de données localisées (%MD<i>, %KD<i>) ou flottantes (%MF<i>, %KF<i>) ne sont pas disponibles.

%Q

Selon le standard IEC, %Q indique un objet langage de type sortie TOR.

%QD

Selon le standard IEC, %MW indique un objet langage de type double mot de sortie.

Seuls les objets d'E/S permettent de localiser les instances de type (%MD<i>, %KD<i>, %QD, %ID, %MF<i>, %KF<i>, %QF, %IF) en utilisant leur adresse topologique (par exemple %MD0.6.0.11, %MF0.6.0.31).

%QF

Selon le standard IEC, %MW indique un objet langage de type réel de sortie.

Seuls les objets d'E/S permettent de localiser les instances de type (%MD<i>, %KD<i>, %QD, %ID, %MF<i>, %KF<i>, %QF, %IF) en utilisant leur adresse topologique (par exemple %MD0.6.0.11, %MF0.6.0.31).

%QW

Selon le standard IEC, %QW indique un objet langage de type sortie analogique.

A

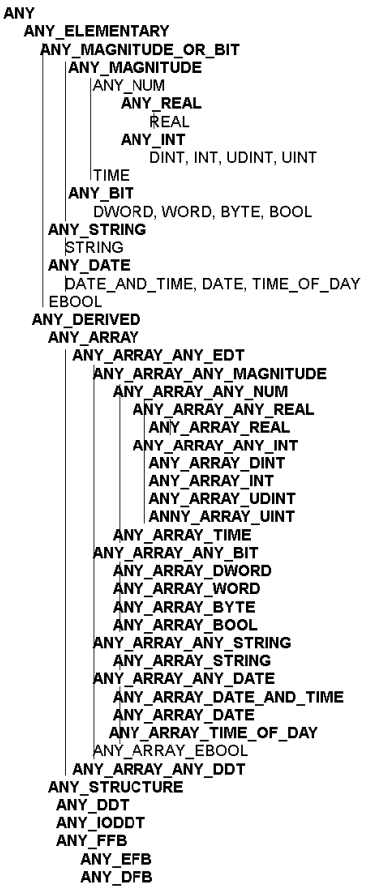
Animation des liens

Egalement appelé **flux des signaux**. Ce terme fait référence à un type d'animation utilisé avec le langage schéma à contacts et les blocs fonction. Les liens apparaissent en rouge, en vert ou en noir selon les variables connectées.

ANY

Une hiérarchie existe entre les différents types de données. Dans les DFB, il est parfois possible de déclarer les variables pouvant contenir plusieurs types de valeurs. On utilise alors les types `ANY_xxx`.

La figure suivante décrit cette structure hiérarchisée :



ARRAY

Un ARRAY est un tableau d'éléments de même type.

La syntaxe est la suivante : ARRAY [<bornes>] OF <Type>

Exemple :

ARRAY [1..2] OF BOOL est un tableau à une dimension composé de deux éléments de type BOOL.

ARRAY [1..10, 1..20] OF INT est un tableau à deux dimensions composé de 10x20 éléments de type INT.

ASCII

Abréviation de American Standard Code for Information Interchange (Code standard américain pour l'échange des données).

Il s'agit d'un code américain (devenu par la suite un standard international) qui utilise 7 bits pour définir chaque caractère alphanumérique utilisé en anglais, les symboles de ponctuation, certains caractères graphiques et d'autres commandes diverses.

B

BCD

Le format BCD (Binary Coded Decimal) est utilisé pour représenter des nombres décimaux compris entre 0 et 9 à l'aide d'un groupe de quatre bits (demi-octet).

Dans ce format, les quatre bits utilisés pour coder les nombres décimaux ont une plage de combinaisons inutilisées.

Exemple de codage BCD :
- Le nombre 2450
- est codé : 0010 0100 0101 0000

BIT

Unité binaire pour une quantité d'informations pouvant représenter deux valeurs distinctes (ou états distincts) : 0 ou 1.

Bloc fonction

Voir EFB

BOOL

BOOL est l'abréviation du type booléen. Il s'agit de l'élément de données de base en informatique. Une variable de type BOOL possède l'une ou l'autre des valeurs suivantes : 0 (FALSE) ou 1 (TRUE).

Un bit extrait de mot est de type BOOL, par exemple : %MW10.4.

BYTE

Lorsque 8 bits sont regroupés, on parle alors de BYTE (octet). La saisie d'un BYTE s'effectue soit en mode binaire, soit en base 8.

Le type BYTE est codé sur un format 8 bits qui, au format hexadécimal, va de 16#00 à 16#FF.

C

Constantes

Variable de type INT, DINT ou REAL localisée en zone constante (%K) ou variables utilisées pour l'adressage direct (%KW, %KD ou %KF). Leur contenu ne peut pas être modifié par le programme en cours d'exécution.

Convention de désignation (identificateur)

Un identificateur est une suite de lettres, de chiffres et de caractères de soulignement commençant par une lettre ou un caractère de soulignement (par exemple, le nom d'un type de bloc fonction, une instance, une variable ou une section). Les lettres des jeux de caractères nationaux (comme ö, ü, é et õ) peuvent être utilisées, sauf dans les noms de projet et de DFB. Les caractères de soulignement sont significatifs dans les identificateurs ; par exemple, A_BCD et AB_CD sont interprétés comme des identificateurs différents. Plusieurs caractères de soulignement au début ou consécutifs ne sont pas valides.

Les identificateurs ne peuvent pas contenir d'espace. Il ne distinguent pas les majuscules des minuscules. Par exemple, ABCD et abcd sont interprétés comme un seul et même identificateur.

Selon le standard IEC 61131-3, les chiffres non significatifs ne sont pas autorisés dans les identificateurs. Cependant, vous pouvez les utiliser si vous cochez dans la boîte de dialogue **Outils →Options du projet** (sous l'onglet **Extensions de langage)** la case **Chiffres non significatifs**.

Les identificateurs ne peuvent pas être des mots clés.

CPU

Abréviation de Control Processing Unit (Unité de traitement des commandes).

Il s'agit du microprocesseur. Il se compose de l'unité de commande et de l'unité arithmétique. L'objectif de l'unité de commande est de récupérer l'instruction à exécuter et les données requises pour exécuter cette instruction dans la mémoire centrale, d'établir des raccordements électriques dans l'unité arithmétique et la logique et d'exécuter le traitement de ces données dans cette unité. Les mémoires ROM ou RAM sont parfois incluses dans la même puce, ainsi que les interfaces E/S ou les mémoires tampon.

D

DATE

Le type DATE codé en BCD sur un format de 32 bits contient les informations suivantes :

- l'année codée dans un champ de 16 bits ;
- le mois codé dans un champ de 8 bits ;
- le jour codé dans un champ de 8 bits.

Le type DATE doit être saisi comme suit : **D#** <Année> **-** <Mois> **-** <Jour>

Ce tableau donne les butées inférieure/supérieure de chaque champ :

Champ	Butées	Commentaire
Année	[1990,2099]	Année
Mois	[01,12]	Le 0 de gauche est toujours affiché ; il peut être omis lors de la saisie.
Jour	[01,31]	Pour les mois 01/03/05/07/08/10/12
	[01,30]	Pour les mois 04/06/09/11
	[01,29]	Pour le mois 02 (années bissextiles)
	[01,28]	Pour le mois 02 (années non bissextiles)

DATE_AND_TIME

Voir DT

DBCD

Représentation d'un entier double au format double BCD.

Le format BCD (Binary Coded Decimal) est utilisé pour représenter des nombres décimaux compris entre 0 et 9 à l'aide d'un groupe de quatre bits.

Dans ce format, les quatre bits utilisés pour coder les nombres décimaux ont une plage de combinaisons inutilisées.

Exemple de codage DBCD :

- Le nombre `78993016`
- est codé : `0111 1000 1001 1001 0011 0000 0001 0110`

DDT

L'abréviation DDT est utilisée pour Derived Data Type (type de données dérivées).

Un type de données dérivées est un ensemble d'éléments de même type (`ARRAY`) ou de types différents (structure).

DFB

DFB est l'abréviation de Derived Function Block (bloc fonction dérivé).

Les types DFB sont des blocs fonction programmables par l'utilisateur en langage ST, IL, LD ou FBD.

L'utilisation de ces types DFB dans une application permet :

- de simplifier la conception et la saisie du programme ;
- d'accroître la lisibilité du programme ;
- de faciliter sa mise au point ;
- de diminuer le volume du code généré.

DINT

`DINT` est l'abréviation du format Double INTeger (entier double) (codé sur 32 bits).

Les butées inférieure et supérieure sont les suivantes : -(2 puissance 31) à (2 puissance 31) - 1.

Exemple :

`-2147483648`, `2147483647`, `16#FFFFFFFF`.

Documentation

Contient toutes les informations concernant le projet. Une fois compilée, la documentation est imprimée et utilisée à des fins de maintenance.

Les informations incluses dans la documentation comprennent :

- les configurations matérielle et logicielle ;
- le programme ;
- les types DFB ;
- les variables et les tables d'animation ;
- les références croisées.
- ...

Lors de la conception du fichier de documentation, vous pouvez inclure l'ensemble de ces éléments ou seulement certains d'entre eux.

DT

DT est l'abréviation de Date and Time.

Le type DT, codé en BCD sur un format de 64 bits, contient les informations suivantes :

- l'année codée dans un champ de 16 bits ;
- le mois codé dans un champ de 8 bits ;
- le jour codé dans un champ de 8 bits ;
- l'heure codée dans un champ de 8 bits ;
- les minutes codées dans un champ de 8 bits ;
- les secondes codées dans un champ de 8 bits.

NOTE : Les 8 bits de poids faible sont inutilisés.

La saisie du type DT est la suivante :

DT# <Année> **-** <Mois> **-** <Jour> **-** <Heure> **:** <Minutes> **:** <Secondes>

Ce tableau donne les butées inférieure/supérieure de chaque champ :

Champ	Butées	Commentaire
Année	[1990,2099]	Année
Mois	[01,12]	Le 0 de gauche est toujours affiché ; il peut être omis lors de la saisie.
Jour	[01,31]	Pour les mois 01/03/05/07/08/10/12
	[01,30]	Pour les mois 04/06/09/11
	[01,29]	Pour le mois 02 (années bissextiles)
	[01,28]	Pour le mois 02 (années non bissextiles)

Champ	Butées	Commentaire
Heure	[00,23]	Le 0 de gauche est toujours affiché ; il peut être omis lors de la saisie.
Minute	[00,59]	Le 0 de gauche est toujours affiché ; il peut être omis lors de la saisie.
Seconde	[00,59]	Le 0 de gauche est toujours affiché ; il peut être omis lors de la saisie.

DWORD

DWORD est l'abréviation de Double Word (mot double).

Le type DWORD est codé sur un format de 32 bits.

Ce tableau donne les butées inférieure/supérieure des bases qui peuvent être utilisées :

Base	Butée inférieure	Butée supérieure
Hexadécimale	16#0	16#FFFFFFFF
Octale	8#0	8#37777777777
Binaire	2#0	2#11111111111111111111111111111111

Exemples de représentation :

Donnée	Représentation dans l'une des bases
00000000000010101101110011011110	16#ADCDE
00000000000000010000000000000000	8#200000
00000000000010101011110011011110	2#10101011110011011110

E

EBOOL

EBOOL est l'abréviation du type Extended BOOLean (booléen étendu). Une variable de type EBOOL présente une valeur (0 (FALSE) ou 1 (TRUE), mais également des fronts montants ou descendants et des fonctions de forçage.

Une variable de type EBOOL occupe un octet de mémoire.

L'octet se compose comme suit :

- un bit pour la valeur ;
- un bit pour le bit d'historique (chaque fois que l'objet d'état change, la valeur est copiée dans le bit d'historique),
- un bit pour le forçage (égal à 0 si l'objet n'est pas forcé, égal à 1 si le bit est forcé).

La valeur par défaut de chaque bit est 0 (FALSE).

Ecran d'exploitation

Editeur intégré à Unity Pro et utilisé pour faciliter le fonctionnement d'un processus automatisé. L'utilisateur contrôle et surveille l'opération d'installation et, en cas de problème, peut intervenir rapidement et simplement.

EDT

EDT est l'abréviation de Elementary Data Type (type de données de base).

Les types EDT disponibles sont les suivants :

- BOOL ;
- EBOOL ;
- WORD ;
- DWORD ;
- INT ;
- DINT ;
- UINT ;
- UDINT ;
- REAL ;
- DATE ;
- TOD ;
- DT.

EF

Abréviation de Elementary Function (fonction élémentaire).

Il s'agit d'un bloc, utilisé dans un programme, qui réalise une fonction logicielle prédéfinie.

Une fonction ne dispose pas d'informations sur l'état interne. Plusieurs appels de la même fonction à l'aide des mêmes paramètres d'entrée fournissent toujours les mêmes valeurs de sortie. Vous trouverez des informations sur la forme graphique de l'appel de fonction dans le "[bloc fonctionnel (instance)]". Contrairement à l'appel des blocs fonction, les appels de fonction comportent uniquement une sortie qui n'est pas nommée et dont le nom est identique à celui de la fonction. Dans FBD, chaque appel est indiqué par un [numéro] unique via le bloc graphique. Ce numéro est généré automatiquement et ne peut pas être modifié.

Vous positionnez et paramétrez ces fonctions dans votre programme afin d'exécuter votre application.

Vous pouvez également en développer d'autres à l'aide du kit de développement SDKC.

EFB

Abréviation de Elementary Function Block (bloc fonction élémentaire).

Il s'agit d'un bloc, utilisé dans un programme, qui réalise une fonction logicielle prédéfinie.

Les EFB possèdent des états et des paramètres internes. Même si les entrées sont identiques, les valeurs des sorties peuvent différer. Par exemple, un compteur possède une sortie qui indique que la valeur de présélection est atteinte. Cette sortie est paramétrée sur 1 lorsque la valeur en cours est égale à la valeur de présélection.

EN/ENO (Activation/Notification d'erreur)

EN correspond à **EN**able (activer) ; il s'agit d'une entrée de bloc facultative.

Si EN = 0, le bloc n'est pas activé, son programme interne n'est pas exécuté et ENO est paramétré sur 0.

Si EN = 1, le programme interne du bloc est exécuté et ENO est paramétré sur 1. Si une erreur survient, ENO est paramétré sur 0.

ENO correspond à **E**rror **NO**tification (notification d'erreur) ; il s'agit de la sortie associée à l'entrée facultative EN.

Si ENO est paramétré sur 0 (car EN = 0 ou en cas d'erreur d'exécution) :

- l'état des sorties de blocs fonction reste identique à celui dans lequel elles étaient lors du dernier cycle de scrutation exécuté correctement ;
- les sorties de fonctions, ainsi que les procédures, sont paramétrées sur "0".

NOTE : Si l'entrée EN n'est pas connectée, elle est automatiquement paramétrée sur 1.

Exécution cyclique

La tâche maître est exécutée soit de manière cyclique, soit de manière périodique. L'exécution cyclique consiste à enchaîner les cycles les uns après les autres sans temps d'attente entre eux.

Exécution périodique

La tâche maître est exécutée soit de manière cyclique, soit de manière périodique. En mode périodique, vous déterminez une durée précise (période) pendant laquelle la tâche maître doit être exécutée. Si l'exécution est réalisée dans les délais, un temps d'attente a lieu avant le cycle suivant. Si le temps d'exécution est plus long, un système de contrôle signale ce dépassement. En cas de dépassement trop important, l'automate est arrêté.

F

FBD

Abréviation de Function Block Diagram (langage en blocs fonctionnels).

FBD est un langage de programmation graphique qui fonctionne comme un logigramme. En complément des blocs logiques simples (`ET`, `OU`, etc.), chaque fonction ou bloc fonction du programme est représenté sous cette forme graphique. Pour chaque bloc, les entrées se situent à gauche et les sorties à droite. Les sorties des blocs peuvent être liées aux entrées d'autres blocs afin de former des expressions complexes.

Fenêtre de visualisation

Cette fenêtre, également appelée fenêtre de surveillance, affiche les variables qui ne peuvent pas être animées dans les éditeurs de langage. Seules les variables visibles à un moment donné dans l'éditeur sont disponibles.

FFB

Terme collectif pour EF (fonction élémentaire), EFB (bloc fonction élémentaire) et DFB (bloc fonction dérivé)

Flash Eprom

Carte mémoire PCMCIA contenant le programme et les constantes de l'application.

FNES

Abréviation de Fichiers Neutres d'Entrées Sorties.

Le format FNES décrit, à l'aide d'une arborescence, les automates en termes de rack, de cartes et de voies.

Il est basé sur le standard CNOMO (Comité de Normalisation des Outillages de Machines-Outils).

Fonction

Voir EF

Fonction élémentaire

Voir EF

G

GRAY

Le code Gray ou "binaire réfléchi" permet de coder une valeur numérique développée en chaîne de configurations binaires qui peut être différenciée par le changement d'état d'un seul bit.

Ce code peut être utilisé, par exemple, pour éviter l'événement aléatoire suivant : en binaire pur, le changement de la valeur 0111 en 1000 peut produire des nombres aléatoires compris entre 0 et 1 000, étant donné que les bits ne changent pas tous de valeur simultanément.

Equivalence entre décimal, BCD et Gray :

Décimal	0	1	2	3	4	5	6	7	8	9
BCD	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001
Gray	0000	0001	0011	0010	0110	0111	0101	0100	1100	1101

I

IEC 61131-3

Standard international : commandes de logique programmable

Partie 3 : langages de programmation.

IL

IL est l'abréviation de Instruction List (liste d'instructions).

Ce langage est une suite d'instructions basiques.

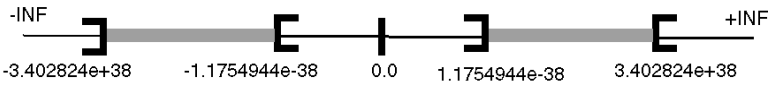
Il est très proche du langage d'assemblage utilisé pour programmer les processeurs.

Chaque instruction est composée d'un code instruction et d'un opérande.

INF

Utilisé pour signifier qu'un nombre dépasse les limites autorisées.

Pour un nombre de type entier, les plages de valeurs (indiquées en gris) sont les suivantes :



Lorsqu'un résultat est :

- inférieur à -3,402824e+38, le symbole `-INF` (pour - infini) est affiché ;
- supérieur à +3,402824e+38, le symbole `+INF` (pour + infini) est affiché.

Instance DFB

Une instance de type DFB se produit lorsqu'une instance est appelée depuis un éditeur de langage.

L'instance possède un nom et des interfaces d'entrée et de sortie ; les variables publiques et privées sont dupliquées (une duplication par instance, le code n'étant pas dupliqué).

Un type DFB peut comporter plusieurs instances.

Instancier

Instancier un objet consiste à allouer un espace en mémoire dont la taille dépend du type de l'objet à instancier. Lorsqu'un objet est instancié, cela prouve qu'il existe et il peut être manipulé par le programme.

INT

INT est l'abréviation du format Single INTeger (entier simple) (codé sur 16 bits).

Les butées inférieure et supérieure sont les suivantes : -(2 puissance 31) à (2 puissance 31) - 1.

Exemple :

-32768, 32767, 2#1111110001001001, 16#9FA4.

IODDT

IODDT est l'abréviation d'Input/Output Derived Data Type (type de données dérivées E/S).

Le terme IODDT désigne un type de données structuré représentant un module ou une voie d'un module automate. Chaque module expert présente ses propres IODDT.

J

Jeton

Etape active d'un SFC.

Jeton unique

Mode de fonctionnement d'un graphe SFC pour lequel une seule étape peut être active à la fois.

L

LD

LD est l'abréviation de Ladder Diagram (langage schéma à contacts).

LD est un langage de programmation, représentant les instructions à exécuter sous forme de schémas graphiques très proches d'un schéma électrique (contacts, bobines, etc.).

Libellé entier

Utilisé pour saisir des valeurs d'entier dans le système décimal. Les valeurs peuvent être précédées d'un signe (+/-). Les caractères de soulignement (_) séparant les nombres ne sont pas significatifs.

Exemple :

`-12, 0, 123_456, +986`

Libellés réels

Nombre exprimé en un ou plusieurs décimaux.

Exemple :

`-12.0, 0.0, +0.456, 3.14159_26`

Libellés réels avec exposant

Nombre pouvant être exprimé à l'aide d'une notation scientifique standard. La représentation est alors la suivante : mantisse + exposant.

Exemple :

`-1,34E-12` ou `-1,34e-12`

`1,0E+6` ou `1,0e+6`

`1,234E6` ou `1,234e6`

Lien hypertexte

La fonction de lien hypertexte permet la création de liens entre votre projet et des documents externes. Vous pouvez créer des liens hypertexte dans tous les éléments du répertoire du projet, dans les variables, dans les objets de l'écran de traitement, etc.

Les documents externes peuvent être des pages Web ou des fichiers (xls, pdf, wav, mp3, jpg, gif, etc.).

M

Macro-étape

Représentation symbolique d'un ensemble unique d'étapes et de transitions, commençant par une étape d'entrée (IN) et se terminant par une étape de sortie (OUT).

Une macro-étape peut appeler une autre macro-étape.

Module fonctionnel

Un module fonctionnel est un regroupement d'éléments de programme (sections, sous-programmes, macro-étapes, tables d'animation, écrans d'exploitation, etc.) destiné à réaliser une fonction d'automatisme.

Il peut être lui-même décomposé en modules fonctionnels de niveau inférieur, ces modules assumant, par rapport à la fonction principale, une ou plusieurs sous-fonctions d'automatisme.

Monotâche

Application comprenant une seule tâche, et nécessairement la tâche maître.

Mot clé

Un mot clé est une combinaison de caractères unique employée en tant qu'élément syntaxique d'un langage de programmation (voir la définition relative au standard IEC 61131-3 fournie à l'annexe B. Tous les mots clés utilisés dans Unity Pro et mentionnés dans le standard IEC 61131-3 sont répertoriés dans l'annexe C de ce standard. Ils ne peuvent pas servir d'identificateurs (noms de variables, sections, types DFM, etc.) dans votre programme).

Multijeton

Mode de fonctionnement d'un SFC. En mode multijeton, le SFC peut posséder plusieurs étapes actives simultanément.

Multitâche

Application constituée de plusieurs tâches (Mast, rapide, auxiliaire, traitement événementiel).

Un ordre de priorité d'exécution des tâches est défini par le système d'exploitation de l'automate.

N

NAN

Utilisé pour signifier qu'un résultat d'opération n'est pas un nombre (NAN = Not A Number).

Exemple : calcul de la racine carrée d'un nombre négatif.

NOTE : le standard IEC 559 définit deux classes de NAN : le NAN silencieux (QNAN) et le NaN de signalisation (SNaN). Un QNAN est un NAN avec un bit de fraction de poids fort tandis qu'un SNaN est un NAN sans bit de fraction de poids fort (numéro de bit 22). Les QNAN peuvent être propagés par le biais de la plupart des opérations arithmétiques sans indication d'une exception. Quant aux SNAN, ils signalent en général une opération non valide lorsqu'ils sont utilisés en tant qu'opérandes dans des opérations arithmétiques (voir %SW17 et %S18).

O

Objet d'E/S

Un objet d'E/S est un objet langage implicite ou explicite pour un module fonction expert ou un équipement d'E/S sur un bus de terrain. Ce sont les types suivants : %Ch, %I, %IW, %ID, %IF, %Q, %QW, % QD, QF, %KW, %KD, %KF, %MW, %MD, et %MF.

L'adresse topologique des objets dépend de la position du module sur le rack ou de la position de l'équipement sur le bus.

Pour les automates Premium/Atrium, les instances de type double de données localisées (%MD<i>, %KD<i>) ou flottantes (%MF<i>, %KF<i>) doivent être localisées par un type d'entier (%MW<i>, %KW<i>). Seuls les objets d'E/S permettent de localiser les instances de type (%MD<i>, %KD<i>, %QD, %ID, %MF<i>, %KF<i>, %QF, %IF) en utilisant leur adresse topologique (par exemple %MD0.6.0.11, %MF0.6.0.31).

Pour les automates Modicon M340, les instances de type double de données localisées (%MD<i>, %KD<i>) ou flottantes (%MF<i>, %KF<i>) ne sont pas disponibles.

Objets SFC

Un objet SFC est une structure de données représentant les propriétés d'état d'une action ou d'une transition d'un graphe séquentiel.

P

Pilote

Programme indiquant au système d'exploitation de votre ordinateur la présence et les caractéristiques d'un périphérique. Nous utilisons également le terme de "pilote de périphérique". Les pilotes les plus connus sont les pilotes d'impression. Pour qu'un automate puisse communiquer avec un ordinateur, des pilotes de communication doivent être installés (Uni-Telway, XIP, Fipway, etc.).

Point d'arrêt

Utilisé en mode "mise au point" de l'application.

Il est unique (un seul à la fois) et, une fois atteint, signale au processeur d'arrêter l'exécution du programme.

Utilisé en mode connecté, il peut être positionné sur l'un des éléments suivants du programme :

- réseau LD ;
- phrase littérale structurée ou liste d'instructions ;
- ligne littérale structurée (mode ligne).

Point de surveillance

Utilisé en mode "mise au point" de l'application.

Permet de synchroniser l'affichage des variables animées avec l'exécution d'un élément de programme (contenant le point de surveillance) afin de connaître leurs valeurs à ce point précis du programme.

Procédure

Les procédures sont techniquement des vues fonctionnelles. L'unique différence avec les fonctions élémentaires réside dans le fait que les procédures peuvent comprendre plus d'une sortie et qu'elles gèrent le type de données `VAR_IN_OUT`. En apparence, les procédures ne sont pas différentes des fonctions élémentaires.

Les procédures sont un supplément au standard IEC 61131-3.

Protection

Option empêchant de lire le contenu d'un élément du programme (protection en lecture), ou d'écrire ou de modifier le contenu de ce type d'élément (protection en lecture/écriture).

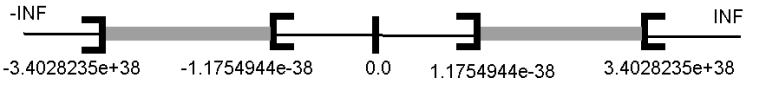
La protection est confirmée par un mot de passe.

R

REAL

Le type REAL (réel) est un type codé sur 32 bits.

Les plages de valeurs possibles sont illustrées en gris dans la figure suivante :



Lorsqu'un résultat est :

- compris entre -1,175494e-38 et 1,175494e-38, il est considéré comme étant un DEN ;
- inférieur à -3,4028234e+38, le symbole -INF (pour - infini) s'affiche ;
- supérieur à +3,4028234e+38, le symbole INF (pour + infini) s'affiche ;
- indéfini (racine carrée d'un nombre négatif), le symbole NAN s'affiche.

NOTE : le standard IEC 559 définit deux classes de NAN : le NAN silencieux (QNAN) et le NaN de signalisation (SNaN). Un QNAN est un NAN avec un bit de fraction de poids fort tandis qu'un SNaN est un NAN sans bit de fraction de poids fort (numéro de bit 22). Les QNAN peuvent être propagés par le biais de la plupart des opérations arithmétiques sans indication d'une exception. Quant aux SNAN, ils signalent en général une opération non valide lorsqu'ils sont utilisés en tant qu'opérandes dans des opérations arithmétiques (voir %SW17 et %S18).

NOTE : Lorsqu'un DEN (nombre non normalisé) est utilisé en tant qu'opérande, le résultat n'est pas significatif.

Réseau

Utilisé principalement pour la communication, un réseau est un groupe de stations qui communiquent entre elles. Le terme "réseau" est également utilisé pour définir un groupe d'éléments graphiques interconnectés. Ce groupe constitue ensuite une partie d'un programme qui peut être composé d'un groupe de réseaux.

RS 232C

Standard de communication de série qui définit la tension du service suivant :

- un signal de +12 V indique un 0 logique ;
- un signal de -12 V indique un 1 logique.

Cependant, en cas d'atténuation du signal, une détection est fournie jusqu'aux limites -3 V et +3 V.

Entre ces deux limites, le signal sera considéré comme non valide.

Les connexions RS 232 sont relativement sensibles aux interférences. Le standard précise de ne pas dépasser une distance de 15 m ou 9 600 bauds (bits/s).

RS 485

Standard de connexion série qui fonctionne dans un différentiel de 10 V/+5 V. Il utilise deux fils pour l'envoi et la réception. Leurs sorties "3 états" leur permettent de passer en mode d'écoute une fois la transmission terminée.

RUN

Fonction permettant de démarrer le programme d'application de l'automate.

RUN Auto

Fonction permettant d'exécuter le démarrage automatique du programme d'application de l'automate en cas de démarrage à froid.

Rung

Un rung est l'équivalent d'une séquence dans le langage schéma à contacts (LD). Les autres termes associés sont "réseau à contacts" ou, plus généralement, "réseau". Un rung est inscrit entre deux barres potentielles d'un éditeur de langage schéma à contacts ; il se compose d'un groupe d'éléments graphiques interconnectés au moyen de connexions horizontales ou verticales. Il est constitué de 17 à 256 lignes et de 11 à 64 colonnes maximum.

S

Section

Module programmable appartenant à une tâche pouvant être écrit dans le langage choisi par le programmeur (FBD, LD, ST, IL ou SFC).

Une tâche peut être composée de plusieurs sections, l'ordre d'exécution des sections au sein de la tâche correspondant à l'ordre dans lequel elles sont créées. Cet ordre peut être modifié.

SFC

Abréviation de Sequential Function Chart (diagramme fonctionnel en séquence).

Le SFC permet de représenter graphiquement et de façon structurée le fonctionnement d'un automatisme séquentiel. Cette description graphique du comportement séquentiel de l'automatisme et des différentes situations qui en découlent s'effectue à l'aide de symboles graphiques simples.

Sous-programmes

Module programmable appartenant à une tâche (Mast, rapide, auxiliaire) pouvant être écrit dans le langage choisi par le programmeur (FBD, LD, ST ou IL).

Un sous-programme ne peut être appelé que par une section ou un autre sous-programme appartenant à la tâche dans laquelle il est déclaré.

ST

Abréviation de Structured Text (langage littéral structuré).

Le langage littéral structuré est un langage élaboré proche des langages de programmation informatiques. Il permet de structurer des suites d'instructions.

STRING

Une variable de type `STRING` est une chaîne de caractères ASCII. La longueur maximale d'une chaîne de caractères est de 65 534 caractères.

Structure

Vue dans le navigateur de projet qui représente la structure du projet.

T

Tâche

Ensemble de sections et de sous-programmes, exécutés de façon cyclique ou périodique pour la tâche MAST, ou périodique pour la tâche rapide.

Une tâche possède un niveau de priorité, et des entrées et des sorties de l'automate lui sont associées. Ces E/S sont actualisées en conséquence.

Tâche maître

Tâche principale du programme.

Elle est obligatoire et est utilisée pour effectuer le traitement séquentiel de l'automate.

Tâche rapide

Tâche déclenchée de façon périodique (réglage de la période dans la configuration du processeur) utilisée pour exécuter une portion d'application de priorité supérieure à la tâche Mast (maître).

Tâches auxiliaires

Tâche périodique optionnelle utilisée pour les processus qui ne nécessitent pas de traitement rapide : mesure, régulation, aide au diagnostic, etc.

TIME

Le type `TIME` exprime une durée en millisecondes. Codé sur 32 bits, ce type permet d'obtenir des durées de 0 à (2 puissance 32) -1 millisecondes.

Time Out

Dans des projets de communication, le Time Out est un délai après lequel la communication est arrêtée en cas d'absence de réponse du périphérique cible.

TIME_OF_DAY

Voir `TOD`

TOD

TOD est l'abréviation de Time Of Day.

Le type TOD, codé en BCD sur un format de 32 bits, contient les informations suivantes :

- l'heure codée dans un champ de 8 bits ;
- les minutes codées dans un champ de 8 bits ;
- les secondes codées dans un champ de 8 bits.

NOTE : Les 8 bits de poids faible sont inutilisés.

La saisie du type Time Of Day est la suivante : **TOD#** <Heure> **:** <Minutes> **:** <Secondes>

Ce tableau donne les butées inférieure/supérieure de chaque champ :

Champ	Butées	Commentaire
Heure	[00,23]	Le 0 de gauche est toujours affiché ; il peut être omis lors de la saisie.
Minute	[00,59]	Le 0 de gauche est toujours affiché ; il peut être omis lors de la saisie.
Seconde	[00,59]	Le 0 de gauche est toujours affiché ; il peut être omis lors de la saisie.

Exemple : TOD#23:59:45.

Traitement événementiel

Le traitement événementiel est une section de programme déclenchée par un événement. Les instructions programmées dans cette section sont exécutées lorsqu'un événement logiciel (temporisation) ou matériel (module métier) est reçu par le processeur.

Les traitements événementiels sont prioritaires par rapport aux autres tâches et sont exécutés dès la détection de l'événement.

Le traitement événementiel EVT0 est prioritaire entre tous. Les autres ont le même niveau de priorité.

NOTE : Pour M340, les événements E/S ayant le même niveau de priorité sont stockés dans un tampon FIFO et sont traités dans l'ordre dans lequel ils ont été reçus.

Tous les temporisateurs ont le même niveau de priorité. Lorsque plusieurs temporisateurs prennent fin simultanément, le plus petit numéro de temporisateur est traité en premier.

Le mot système %SW48 comptabilise le nombre d'événements d'E/S traités.

U

UDINT

UDINT est l'abréviation du format Unsigned Double INTeger (entier double non signé) (codé sur 32 bits). Les butées inférieure et supérieure sont les suivantes : 0 à (2 puissance 32) - 1.

Exemple :

0, 4294967295, 2#11111111111111111111111111111111, 8#37777777777, 16#FFFFFFFF.

UINT

UINT est l'abréviation du format Unsigned INTeger (entier non signé) (codé sur 16 bits). Les butées inférieure et supérieure sont les suivantes : 0 à (2 puissance 16) - 1.

Exemple :

0, 65535, 2#1111111111111111, 8#177777, 16#FFFF.

V

Valeur littérale de temps

Les unités du type TIME sont les suivantes : les jours (d), les heures (h), les minutes (m), les secondes (s) et les millisecondes (ms). Une valeur littérale du type TIME est représentée par une combinaison de types antérieurs précédés de T#, t#, TIME# ou time#.

Exemples : T#25h15m, t#14.7S, TIME#5d10h23m45s3ms

Valeur littérale en base 10

Une valeur littérale en base 10 est utilisée pour représenter une valeur entière décimale. Cette valeur peut être précédée des signes "+" et "-". Si le caractère "_" est utilisé dans cette valeur littérale, il n'est pas significatif.

Exemple :

-12, 0, 123_456, +986

Valeur littérale en base 16

Une valeur littérale en base 16 est utilisée pour représenter un entier hexadécimal. La base est déterminée par le nombre "16" et le signe "#". Les signes "+" et "-" sont interdits. Pour faciliter la lecture, vous pouvez utiliser le signe "_" entre les bits.

Exemple :

`16#F_F` ou `16#FF` (en décimal 255)

`16#F_F` ou `16#FF` (en décimal 224)

Valeur littérale en base 8

Une valeur littérale en base 8 est utilisée pour représenter un entier octal. La base est déterminée par le nombre "8" et le signe "#". Les signes "+" et "-" sont interdits. Pour faciliter la lecture, vous pouvez utiliser le signe "_" entre les bits.

Exemple :

`8#3_77` ou `8#377` (en décimal 255)

`8#34_0` ou `8#340` (en décimal 224)

Valeur littérale en base 2

Une valeur littérale en base 2 est utilisée pour représenter un entier binaire. La base est déterminée par le nombre "2" et le signe "#". Les signes "+" et "-" sont interdits. Pour faciliter la lecture, vous pouvez utiliser le signe "_" entre les bits.

Exemple :

`2#1111_1111` ou `2#11111111` (en décimal 255)

`2#1110_0000` ou `2#11100000` (en décimal 224)

Variable

Entité mémoire du type `BOOL`, `WORD`, `DWORD`, etc., dont le contenu peut être modifié par le programme en cours d'exécution.

Variable localisée

Variable dont la position dans la mémoire de l'automate peut être connue. Par exemple, la variable `Pression_eau` est associée au repère `%MW102`. `Pression_eau` est dite localisée.

Variable non localisée

Variable dont la position dans la mémoire de l'automate ne peut pas être connue. Une variable à laquelle aucune adresse n'a été affectée est dite non localisée.

Vue fonctionnelle

Vue permettant d'afficher la partie du programme de l'application via les modules fonctionnels créés par l'utilisateur (voir la définition relative au module fonctionnel).

W

WORD

Le type `WORD` est codé sur un format de 16 bits et est utilisé pour effectuer des traitements sur des chaînes de bits.

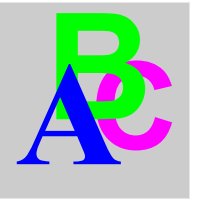
Ce tableau donne les butées inférieure/supérieure des bases qui peuvent être utilisées :

Base	Butée inférieure	Butée supérieure
Hexadécimale	16#0	16#FFFF
Octale	8#0	8#177777
Binaire	2#0	2#1111111111111111

Exemples de représentation

Donnée	Représentation dans l'une des bases
0000000011010011	16#D3
1010101010101010	8#125252
0000000011010011	2#11010011

Index



Symbols

A

B

C

D

M

N

O

P

R

S

T

U

V

W

X